

In [1]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))
```

In [2]:

```
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 sklearn.naive_bayes import GaussianNB
from datetime import datetime
from tqdm import tqdm
from sklearn.calibration import CalibratedClassifierCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.externals import joblib
%autosave 180
```

Autosaving every 180 seconds

MPST Movie Plot Synopses Tag Prediction : Tag Prediction

1. Business Problem

1.1 Description

Social tagging of movies reveals a wide range of heterogeneous information about movies, like the genre, plot structure, soundtracks, metadata, visual and emotional experiences. Such information can be valuable in building automatic systems to create tags for movies. Automatic tagging systems can help 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 paper, we set 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 around 70 tags exposing heterogeneous characteristics of movie plots and the multi-label associations of these tags with some 14K movie plot synopses. We investigate how 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 inferring tags from plot synopses. We expect the corpus will be useful in other tasks where analysis of narratives is relevant.

Credit: <https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags> (<https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags>)

Problem Statement

Suggest the tags based on the movie plots that was there in the given dataset

1.2 Source / useful links

Data Source : <https://www.kaggle.com/criptexcode/mpst-movie-plot-synopses-with-tags> (<https://www.kaggle.com/criptexcode/mpst-movie-plot-synopses-with-tags>)

Research paper : <https://www.aclweb.org/anthology/L18-1274> (<https://www.aclweb.org/anthology/L18-1274>)

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 on Movie Sites
3. No strict latency constraints.

2. Machine Learning problem

2.1 Data

2.1.1 Data Overview

Refer: <https://www.kaggle.com/criptexcode/mpst-movie-plot-synopses-with-tags> (<https://www.kaggle.com/criptexcode/mpst-movie-plot-synopses-with-tags>)

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 instance resides in the Train/Validation/Test split.

Data Field Explanation

Dataset contains 14828 rows. The columns in the table are:

IMDB id - Unique identifier which contains the IMDB id of the movie

Title - Contains unique title for each movie

plot_synopsis - Plot Synopsis of the movie tags

Tags - Tags assigned to each of the movie

split - Position of the movie in the standard data split, indicates whether a data point belongs to train, test or validation data

synopsis_source - Source from where the plot synopsis for each movie was collected

2.1.2 Example Data point

Title: A Single Man

Plot Synopses : George Falconer (Colin Firth) approaches a car accident in the middle of a snow-white scenery. There is a bloodied man there and he kisses him. He wakes up: he was dreaming about the moment when his partner of 16 years, Jim (Mathew Goode), died--though he was not there with him because Jim was visiting his disapproving family on his own. George remembers the phone ringing on that fateful day, when Jim's cousin told him about the fatal accident, and how George was not welcome to attend the funeral, because of the family's homophobia (common for the period and later). George remembers breaking down to Charley (Julianne Moore) that day, his best friend from his life in London, who had also relocated to LA; once briefly sexually attached to George before he was completely honest with himself, she may still feel attracted to him. George showers and dresses. It's November 30, 1962, the eve of the Cuban missile crisis. Though British, he is now a professor of English at UCLA. He is depressed, never having recovered from his loss; and when he leaves for work, he packs a gun in his briefcase. He tells his cleaning lady Alva (Paulette Goddard) that she has always been wonderful - in spite of her having forgotten to take out the bread from the fridge. George hugs her, which leaves her utterly confused. On campus, George notices a couple of students, chain-smoking Lois (Nicole Steinwedell) and a boy. One of the secretaries (Keri Lynn Pratt) tells him that she has given his address to some nice new student; it turns out to be this boy, Kenny Potter (Nicholas Hoult), who talks to him after class about the speech George has just given out in the classroom concerning minorities and fear. Kenny discusses recreational drug use with George who tells him that he had never heard George express himself so openly in class as he had that day. He buys George a pencil sharpener as a token of gratitude for George's talking with him. George phones Charley, who is dressing for the dinner they have planned at her home. George gets into his car, and picks his gun after having cleaned up his office. However, Kenny appears once again, and invites him to go for a drink, observing George's depression and having noticed that he has cleaned out the desk in his office. George tells him it will have to be some other time. He goes to the bank to pick up various things from his safe deposit box, and when looking at a photo of his deceased lover, recalls a conversation with him on the beach. After buying some bullets, he goes to a convenience store. There, Carlos (Jon Kortajarena) bumps onto him, breaking the bottle of Scotch he has just bought. George buys a new bottle of Scotch and they talk. They smoke a few cigarettes and drink a bottle of gin together. George leaves, refusing Carlos' offer of company, saying that this is a serious day for him and that he's trying to get over an old love. At home, he puts on a record and remembers a conversation with Jim while each one was reading a different book on a couch. He pretends shooting himself as practice for later that night, but in a semi-comic scene, can't find the best position in which to accomplish it. Charley calls to remind him of their dinner plans, which he grudgingly attends after leaving a note and some money for Alva. They dance and talk about London, life, Charley's ex-husband's abandonment, and she offers George by suggesting that they might have had a "normal" life together if he hadn't been a "poof." Charley says George doesn't look well, reminding him of the heart attack he suffered near the time of Jim's death. Charley tries to convince George to spend the night at her home, but he leaves. The scene flashes back to 1946 when Jim and George had met when at a bar. Jim was on leave from the Army, right after the second world war. Returning to 1962, we see George returning to the same bar, near his home; now a quiet place where he asks for a Scotch. Kenny has followed him there. They talk and then go to the beach and swim naked. They go to George's place. As George's forehead is bleeding, Kenny tends to it, and sees in the medicine's cabinet a nude photo of Jim. George sees Kenny strip off his wet clothes, but does nothing. Kenny says that he and Lois are not romantically involved. Not unlike George and Charley in the distant past, Kenny explains that they had a brief sexual liaison. Kenny and George do not have sex, and Kenny stays on the couch, given the very late hour. George wakes in a few hours, and finds his gun under Kenny's covers and removes it, locking it up as Kenny sleeps. When he returns to bed, George dies of a heart attack, seeing the image of Jim kissing his forehead.

Tags : 'gothic, cruelty, violence, cult, revenge, sadist'

2.2 Mapping the real-world problem to a Machine Learning Problem

2.2.1 Type of Machine Learning Problem

It is a multi-label classification problem

Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A movie plot synopsis may either have tags like horror, sad, violence, brutal or it may have all of these 4 tags.

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

2.2.2 Performance metric

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 reaches its best 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 class imbalance.

'Macro f1 score':

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

<https://www.kaggle.com/wiki/MeanFScore> (<https://www.kaggle.com/wiki/MeanFScore>)
http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (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://www.kaggle.com/wiki/HammingLoss> (<https://www.kaggle.com/wiki/HammingLoss>)

3. Exploratory Data Analysis

3.1 Data Loading

In [3]:

```
#Load the pandas dataframe, display the number of columns, display the first five rows
data=pd.read_csv("mpst_full_data.csv")
print("Columns present in the data: ",[i for i in data.columns])
print("Number of data points: ",data.shape[0])
data.head()
```

Columns present in the data: ['imdb_id', 'title', 'plot_synopsis', 'tags', 'split', 'synopsis_source']
Number of data points: 14828

Out[3]:

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
0	tt0057603	I tre volti della paura	Note: this synopsis is for the original Italian...	cult, horror, gothic, murder, atmospheric	train	imdb
1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb
2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb
3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb

3.2 Creating a SQL db file from the given csv file

In [4]:

```
#Learn SQL: https://www.w3schools.com/sql/default.asp
if not os.path.isfile('train.db'):
    start = datetime.now()
    disk_engine = create_engine('sqlite:///train.db')
    start = dt.datetime.now()
    chunksize = 15000
    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
        df.to_sql('data', disk_engine, if_exists='append')
        index_start = df.index[-1] + 1
print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:00:11.771431

3.3 Counting the number of rows

In [25]:

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    num_rows = pd.read_sql_query("""SELECT count(*) FROM data""", con)
    print("Number of rows in the database :", num_rows['count(*)'].values[0])
    con.close() #Always remember to close the database
    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 : 0:00:00.788836

3.4 Check for the distribution of train, validation and test data points in the given dataset

In [26]:

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    split_info = pd.read_sql_query('SELECT split as Data_Type, COUNT(*) AS Number_of_Instances FROM data GROUP BY Data_Type', con)
    con.close()

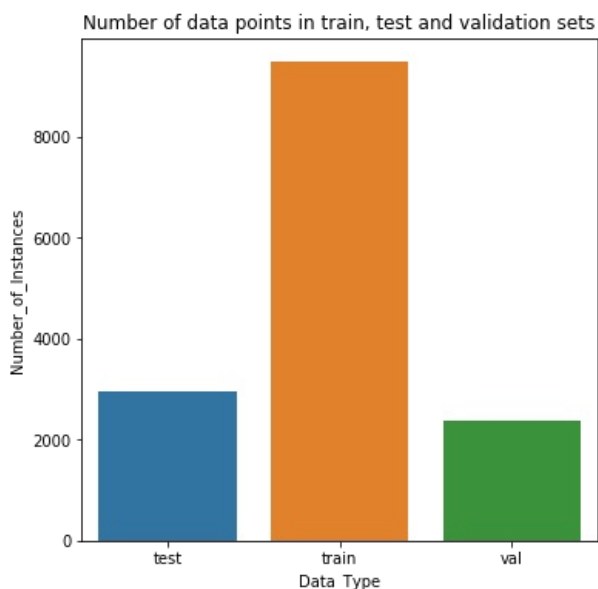
split_info.head()
```

Out[26]:

	Data_Type	Number_of_Instances
0	test	2966
1	train	9489
2	val	2373

In [6]:

```
plt.figure(figsize=(6, 6))
plt.title("Number of data points in train, test and validation sets")
sns.barplot(split_info['Data_Type'], split_info['Number_of_Instances'])
plt.show()
```



Here, we see that the dataset contains 9489 points for training set, 2373 points for test set and 2966 points for test set.

3.5 Check for the distribution of data sources

In [27]:

```
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    data_source = pd.read_sql_query('SELECT synopsis_source as Data_Source, COUNT(*) AS Number_of_data_points FROM data GROUP BY synopsis_source', con)
    con.close()

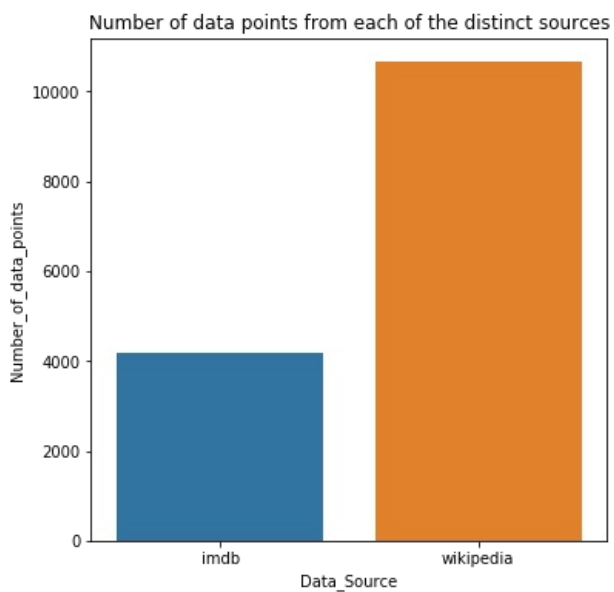
data_source.head()
```

Out[27]:

	Data_Source	Number_of_data_points
0	imdb	4172
1	wikipedia	10656

In [28]:

```
plt.figure(figsize=(6, 6))
plt.title ("Number of data points from each of the distinct sources")
sns.barplot(data_source['Data_Source'],data_source['Number_of_data_points'])
plt.show()
```



We can see that 4172 movie plots are taken from IMDB and 10656 movie plots are taken from Wikipedia.

3.6 Checking for duplicates entries in the dataset

In [29]:

```
#Learn SQL: https://www.w3schools.com/sql/default.asp
if os.path.isfile('train.db'):
    start = datetime.now()
    con = sqlite3.connect('train.db')
    df_no_dup = pd.read_sql_query('SELECT title, plot_synopsis, tags, split, COUNT(*) as cnt_dup FROM data GROUP BY title, plot_synopsis, tags, split', con)
    con.close()
    print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:00:00.473689

In [30]:

```
df_no_dup.head()
```

Out[30]:

	title	plot_synopsis	tags	split	cnt_dup
0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	1
1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	1
2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	1
3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	1
4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	1

In [31]:

```
print("Number of rows in the original dataset: ",num_rows['count(*)'].values[0])
print("Number of rows in the de-duplicated dataset: ",df_no_dup.shape[0])
```

Number of rows in the original dataset: 14828

Number of rows in the de-duplicated dataset: 14781

In [32]:

```
print("Total number of duplicate entries removed from the given dataset: ",num_rows['count(*)'].values[0]-df_no_d
up.shape[0])
print("Percentage of duplicate entries that were originally present: {} %".format(np.round((num_rows['count(*)'].
values[0]-df_no_dup.shape[0])/num_rows['count(*)'].values[0]*100,2)))
```

Total number of duplicate entries removed from the given dataset: 47

Percentage of duplicate entries that were originally present: 0.32 %

3.7 Checking for the number of times each movie has appeared in the database

In [33]:

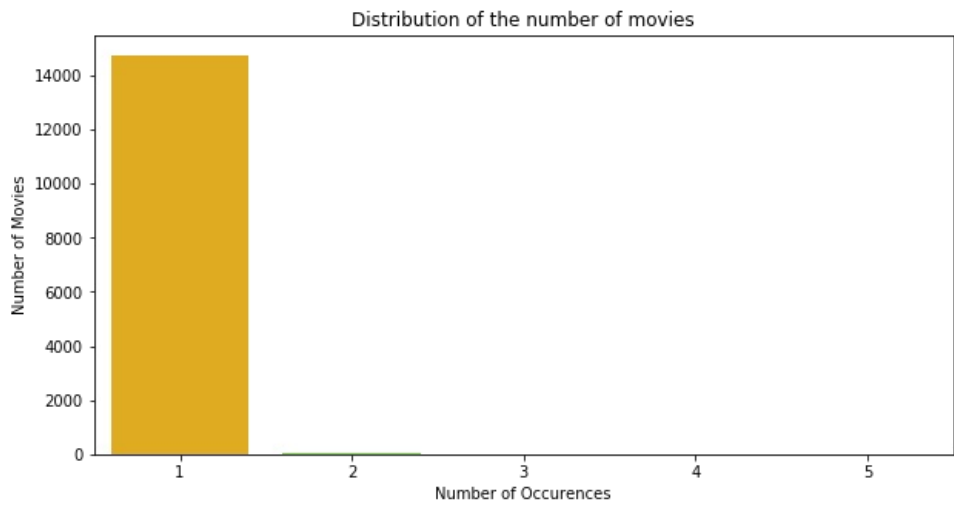
```
#Number of times each movie appeared in the database
#14743 movies occurred only 1 time. 32 occurs 2 times. 4 movies occurs 3 times and 1 movie occurs 5 times
df_no_dup.cnt_dup.value_counts()
```

Out[33]:

```
1    14743
2      32
3       4
5        1
4        1
Name: cnt_dup, dtype: int64
```

In [34]:

```
#Plot the result in a count plot
plt.figure(figsize=(10,5))
sns.countplot(df_no_dup.cnt_dup, palette='gist_rainbow')
plt.title("Distribution of the number of movies")
plt.xlabel("Number of Occurences")
plt.ylabel("Number of Movies")
plt.show()
```



3.8 Checking for the distribution of number of tags per movie.

In [35]:

```
#Here we will add a new feature called tags count, which will count the number of tags in each movie
start = datetime.now()
df_no_dup["tag_count"] = df_no_dup["tags"].apply(lambda text: len(str(text).split(" ")))
print("Time taken to run this cell :", datetime.now() - start)
df_no_dup.head()
```

Time taken to run this cell : 0:00:00.024752

Out[35]:

	title	plot_synopsis	tags	split	cnt_dup	tag_count
0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	1	1
1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	1	1
2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	1	5
3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	1	2
4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	1	7

In [36]:

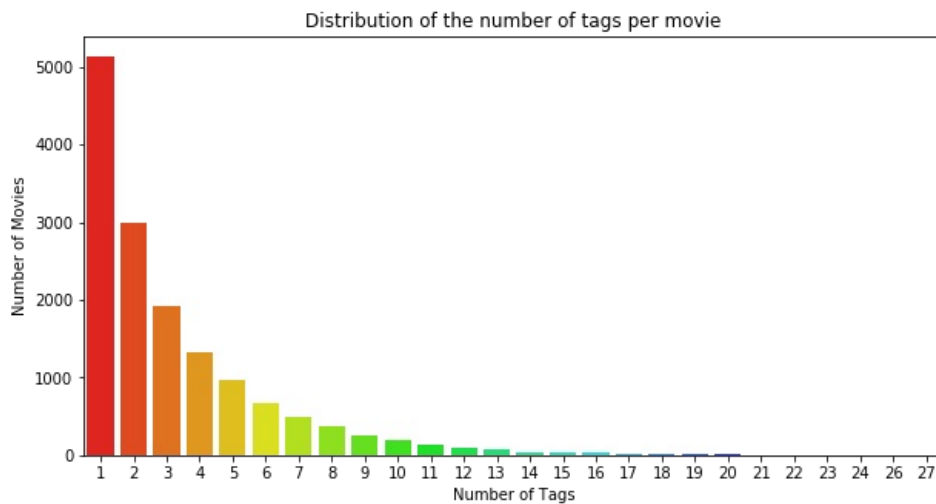
```
#Distribution of number of tags per movie.  
#5133 movies has 1 tag, 2990 has 2 tags, 1924 movies has 3 tags, 1318 movies has 4 tags and so on. There is one movie with 24 tags and 2 movies with 23tags.  
df_no_dup.tag_count.value_counts()
```

Out[36]:

```
1      5133  
2      2990  
3      1924  
4      1318  
5       960  
6       661  
7       492  
8       374  
9       252  
10      191  
11      136  
12      100  
13       70  
14       44  
15       35  
16       31  
17       21  
18       18  
20       11  
19        8  
21        3  
26        2  
27        2  
22        2  
23        2  
24        1  
Name: tag_count, dtype: int64
```

In [37]:

```
#Plot the result in a count plot  
plt.figure(figsize=(10,5))  
sns.countplot(df_no_dup.tag_count, palette='gist_rainbow')  
plt.title("Distribution of the number of tags per movie")  
plt.xlabel("Number of Tags")  
plt.ylabel("Number of Movies")  
plt.show()
```



4. Create a new database with no duplicate entries

In [38]:

```
#Creating a new database with no duplicates  
if not os.path.isfile('train_no_dup.db'):  
    disk_dup = create_engine("sqlite:///train_no_dup.db")  
    no_dup = pd.DataFrame(df_no_dup, columns=['title', 'plot_synopsis', 'tags', 'split'])  
    no_dup.to_sql('no_dup_train', disk_dup)
```

In [39]:

```
#This method seems more appropriate to work with this much data. Creating the connection with database file.
if os.path.isfile('train_no_dup.db'):
    start = datetime.now()
    con = sqlite3.connect('train_no_dup.db')
    tag_data = pd.read_sql_query("""SELECT Tags FROM no_dup_train""", con)
    #Always remember to close the database
    con.close()

    # Let's now drop unwanted column.
    tag_data.drop(tag_data.index[0], inplace=True)
    #Printing first 5 columns from our data frame
    print("Time taken to run this cell :", datetime.now() - start)
else:
    print("Please download the train.db file from drive or run the above cells to generate train.db file")

tag_data.head()
```

Time taken to run this cell : 0:00:06.746424

Out[39]:

	tags
1	flashback
2	suspenseful, neo noir, murder, violence
3	cult, violence
4	murder, anti war, violence, flashback, tragedy...
5	murder

Observations from the above analysis.

- 1. There were almost 0.32% movies which were duplicates. So the first thing we did, is remove the duplicate movies from the actual dataset and save it in a new dataset.
- 2. 14743 movies occurred only 1 time. 32 occurs 2 times. 4 movies occurs 3 times and 1 movie occurs 5 times
- 3. There are movies which contains just 1 tag and there are also movies which contains as many as 24 tags!

5. Analysis of Title Texts and Movie Plot Text

In [4]:

```
#Load the de-duplicated dataset
start = datetime.now()
con = sqlite3.connect('train_no_dup.db')
dataframe = pd.read_sql_query("""SELECT * FROM no_dup_train""", con)
con.close()

dataframe.head()
```

Out[4]:

	index	title	plot_synopsis	tags	split
0	0	\$ Set in Hamburg, West Germany, several criminal...		murder	test
1	1	\$windle A 6th grader named Griffin Bing decides to gat...		flashback	train
2	2	'71 Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence		train
3	3	'A' gai wak Sergeant Dragon Ma (Jackie Chan) is part of th...		cult, violence	train
4	4	'Breaker' Morant In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...		train

In [22]:

```
#Printing some random movie plots from the deduplicated dataset.
sent_1 = dataframe['plot_synopsis'].values[0]
print(sent_1)
print("\nTags: {}".format(dataframe['tags'].values[0]))
print("=="*215)

sent_2 = dataframe['plot_synopsis'].values[1000]
print(sent_2)
print("\nTags: {}".format(dataframe['tags'].values[1000]))
print("=="*215)

sent_3 = dataframe['plot_synopsis'].values[1500]
print(sent_3)
print("\nTags: {}".format(dataframe['tags'].values[1500]))
print("=="*215)

sent_4 = dataframe['plot_synopsis'].values[4900]
print(sent_4)
print("\nTags: {}".format(dataframe['tags'].values[4900]))
print("=="*215)
```

Set in Hamburg, West Germany, several criminals take advantage of the German privacy bank laws to use safe deposit boxes in a German bank to store large amounts of illicit cash. These include a Las Vegas mobster known only as the Attorney (Robert Webber) as well as a ruthless drug smuggler known as the Candy Man (Arthur Brauss) and a crooked overbearing U.S. Army sergeant (Scott Brady) and his meek-mannered partner the Major (Robert Stiles), who conspire on a big heroin and LSD smuggling score. Joe Collins (Warren Beatty), an American bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than \$1.5 million) with the help of Dawn Divine (Goldie Hawn), a hooker with a heart of gold. On the day of the robbery, Joe has Dawn phone in a bomb threat to the bank president, Mr. Kessel (Gert Fröbe), to create a diversion. Joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. The bank is closed and evacuated while Joe uses duplicate keys to empty the criminals' three safe deposit boxes into Dawn's large-size deposit box. (It is implied that Joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with Dawn.) Despite the fact that Kessel insists on burning through the wall to rescue Joe instead of waiting for the time lock to open, Joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. The next day, the three criminals, one by one, discover that their boxes are empty and they cannot complete their schemes or go to the police to report the thief. The Attorney flees the country while the others (Sarge, his partner the Major, and the Candy Man) search Dawn Divine's apartment as she was their common link and find clues that connect her to Joe. Sarge calls Kessel to get Joe's home address, but Joe is quickly tipped off by Kessel and he hurriedly sends Dawn to the train station with a suitcase packed with her take (\$765,000) promising to meet her later someplace out of the country. A long climactic chase begins as Dawn gives the Major the slip at the train station while the Candy Man and the Sarge chase Joe across a rail yard and through the Elbe Tunnel. Joe escapes on a car carrier truck, lugging his suitcase, but the Candy Man and the Sarge follow and catch up in the morning at a frozen lake in the countryside, where the Candy Man crashes a car through the ice and drowns while attempting to run Joe down with a stolen car. Joe escapes again by hopping a train, but during the night the Sarge catches up to him only to find that Joe's suitcase contains nothing but a bottle of champagne and wads of newspaper. They conclude that Dawn double-crossed Joe by repacking the suitcases while he was getting the car, and the Sarge proposes a plan to Joe to go after Dawn together. But, upon drinking a swallow of the champagne, the Sarge instantly goes into violent convulsions and falls down dead. The bottle was one of three that the Candy Man had filled with a solution of concentrated LSD to sneak them through customs earlier in the film. Joe then disembarks from the train and walks away, apparently betrayed by Dawn. An epilog shows Dawn in a sunny climate in the USA, joyfully driving a gleaming new yellow Corvette, and then later cuddling in bed with an unseen someone. The other suitcase is sitting near the bed, and Joe's bomber jacket hangs on the coat rack. Dawn smugly explains to the person she was certain the criminals wouldn't kill him and leave themselves with no way to get the money.

Tags: murder

Years ago, a mob boss named Lucio Malatesta (George Touliatos) pinned the murder of rival Sammy Carbone (Gino Marrocco) on another rival named Angelo Allieghieri (Anthony Quinn), which led to Sammy's son Gianni vowing revenge. Frankie Delano (Sylvester Stallone) has spent his life safeguarding Angelo as well as Angelo's daughter, Jennifer Barrett (Madeleine Stowe), whose unsavory husband Kip Barrett (Harry Van Gorkum) has had their young son Rawley (Ezra Perlman) placed in a boarding school against Jennifer's wishes. Jennifer was raised by her adoptive parents Whitney Towers (John Gilbert) and Peggy Towers (Dawn Greenthalgh) and is not aware that Angelo is her father. After Angelo is killed in a restaurant by a hit man named Bruno (Billy Gardell), Frankie introduces himself, tells Jennifer who he is and what he has been doing. A neurotic mess, Jennifer can barely handle the news that Kip is a philanderer, let alone the revelation that she is a gangster's daughter. But a DVD prepared by Angelo in the case of just such an event convinces Jennifer that it's the truth. Jennifer certainly doesn't want a full-time bodyguard, even Frankie. She ditches Kip and then falls for Italian romance novelist Marcello (Raoul Bova), who lectures at her book club. Frankie has suspicions about Marcello, but his job is to stay on the sidelines. Frankie rescues Jennifer from a string of attacks. With many of Angelo's enemies, including Lucio Malatesta, terminated, Frankie allows her to visit Italy with Marcello. But it turns out that Marcello

is actually Gianni Carboni, who had Angelo killed. And now Gianni plans to kill Jennifer. It's up to Frankie to protect her one more time.

Tags: violence, comedy, murder, flashback

=====

Nick Conklin (Michael Douglas) is a skilled motorcyclist and a tough veteran New York City police officer facing possible criminal charges; Internal Affairs believes Nick was involved with his partner who was caught taking criminal money in a corruption scandal. Nick is divorced from his wife, who has custody of their two children. Nick also has financial difficulties due to alimony and child support as well as other concerns. Nick reports to a criminal investigation hearing being run by two officers from Internal Affairs, a conference that doesn't go well for him. They ask Nick about his involvement with several officers under investigation. When Nick refuses to squeal on his comrades, Internal Affairs threatens him, suggesting he's as corrupt as the others in the department. While having a drink at a local Italian restaurant/bar, Nick and his partner Charlie Vincent (Andy Garcia) observe two Japanese men having what appears to be a friendly lunch with some Italian gangsters. Nick is increasingly suspicious of the group until another Japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the Japanese. As the man turns to leave, one of the Japanese men at the table says, in Japanese, "The Oyabun [Godfather] will not stand for this." The leader of the Japanese group chimes in, "As always, such a troublesome child." The Japanese man finds these remarks insulting and he slashes the man's throat, stabs another in the chest, and then walks out. Nick and Charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills Nick in a nearby slaughterhouse. The suspect turns out to be a Yakuza gangster by the name of Sato (Yusaku Matsuda). The situation is further complicated when Nick's superior officer, Captain Oliver (John Spencer), tells him that Sato is to be extradited to Osaka and given to the police there. Nick is angry that Sato will not be tried for murder in the United States, but agrees to escort him to Japan. Nick's captain also has an ulterior motive for sending Nick overseas; he believes the excursion will keep Nick from causing more trouble and exacerbating the already biased Internal Affairs investigation of him. On the plane, Nick and Charlie talk about Nick's situation and how Nick's own expenses are beyond his means to pay them. At one point, while Charlie is out of his seat, Sato notices Nick cheating at solitaire and contemptibly chuckles to himself. Nick cruelly hits his prisoner in the mouth and lies about it when Charlie returns and asks what happened. When they arrive in Osaka, men identifying themselves as Japanese police immediately meet them on the plane, display a "transfer document" printed in Japanese and take Sato into their custody, leaving the plane by the rear exit. As Nick and Charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in English, indicating that the first "cops" were impostors. Nick and Charlie are taken to the headquarters of the Osaka Prefecture of Police and questioned. They are also blamed for Sato's escape. After much haranguing by Nick (who shows xenophobia) towards the Japanese, who rarely acknowledge that they can speak English, he and Charlie are allowed to observe the hunt for Sato. However, the senior police officer emphasizes that they have no authority in Japan and it is illegal for them to carry their guns, which are confiscated. They are assigned to Masahiro Matsumoto (Takakura), a mild-mannered and experienced officer, who will be their guide. Throughout the investigation Nick behaves rudely, offending Matsumoto, while Charlie tries to be more polite. Taken to a murder scene at a local nightclub, Nick recognizes the murder victim as one of the men at the airport who took Sato into custody. While the dead man is examined by forensic experts, one of them removes a \$100 bill from his mouth. Nick makes contact with an American blond nightclub hostess, Joyce (Kate Capshaw), who explains that the Japanese public, including the giggling hostesses in the club, all believe that Nick and Charlie are not to be taken seriously because they allowed Sato to easily escape from custody, and represent American inefficiency and stupidity. Through Joyce, Nick discovers that Sato is fighting a gang war with a notorious crime boss, Sugai (Tom Isaburo Wakayama). Sato used to be a lieutenant for Sugai and now wants his own territory to rule. Sato had traveled to New York to disrupt a meeting with American Italian gangsters about a scheme being set up by Sugai involving the package Sato had taken in the restaurant. Having joined a police raid of a gang hideout without permission, Nick takes some \$100 bills from a table, which he later shows are forgeries by burning one. The next day Matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to America; Nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in Matsumoto's superior's office. He suggests that the package stolen by Sato was either more samples of the forged bills or plates to make more. Late one night, after spending a few hours in a nightclub with Matsumoto, Nick and Charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from Matsumoto. They are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals Charlie's raincoat and lures Charlie into an underground parking garage. Nick follows, shouting for Charlie to come back, but is separated from his partner by a security gate. The unarmed Nick then watches in horror as Sato and several of his bszoku gang members briefly torture Charlie using swords and knives, before Sato beheads him. Distraught, Nick is comforted by Joyce at her apartment. Matsumoto arrives with Charlie's belongings, including his NYPD badge, which Nick gives to Matsumoto, and Charlie's service pistol, which Nick keeps for himself. Matsumoto and Nick trail one of Sato's operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and Nick admits to Matsumoto that he had taken some money in New York, where he says there is no "black-and-white" procedure, only "gray" areas. Matsumoto disagrees, saying "theft is theft" and that Nick's illegal action disgraces all police, including Matsumoto and Charlie. Nick realizes Matsumoto is right and humbly accepts Matsumoto's advice. In the morning the woman retrieves from a bank strongbox a sample counterfeit note (printed only on one side) which she passes to one of Sato's gang on the street. Nick and Matsumoto tail the man to a steel foundry where they find Sato meeting with Sugai, and discover that the package that Sato had stolen in New York contains one of the printing plates for the American \$100 bill. Nick intervenes when Sato leaves the meeting and a gunfight ensues. Sato escapes again when Nick is arrested by the swarming police for using a gun in public, and told he will be sent back to New York in disgrace. Nick boards the plane for New York but is able to sneak off to pursue Sato on his own. He finds that Matsumoto has been suspended and demoted by his police force, a deep humiliation. Joyce helps him meet Sugai, who explains that making counterfeit U.S. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of Hiroshima and the loss of dignity he and his family faced in the aftermath of W

orld War II. Nick suggests a deal where Sugai can use Nick as an insignificant American to retrieve the stolen plate from Sato, leaving Sugai's reputation and hands clean.Sugai drops Nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. Nick is supplied with a shotgun. Sato arrives a short time later, as does Matsumoto. Matsumoto and Nick discover that Sato's men are planning a massacre. At the meeting table, Sato surrenders his single plate and requests recognition and his own territory. However, Sugai demands that Sato first atone for his offenses against the Yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. As he takes his position next to Sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between Sugai's and Sato's men. Sato escapes the fight on a dirt bike with Nick close behind. Nick is able to spill Sato off his bike and the two fight briefly, until Nick gains the advantage. The scene ends with Nick having to decide whether or not to kill Sato for Charlie and for all the humiliation he has suffered.The film ends with Matsumoto and Nick walking a handcuffed Sato into police HQ to the amazement of everyone and later receiving commendations, which Nick graciously accepts. At the airport, Nick thanks Matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. Underneath the shirt, Matsumoto finds the two counterfeit printing plates.

Tags: boring, neo noir, murder, violence, cult, romantic, suspenseful

The film introduces a circle of youths who are addicted to playing Hellworld, an online computer game based on the Hellraiser series. The film opens at the funeral of Adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. The remaining five friends blame themselves for not having prevented Adam's suicide. Two years later, they attend a private Hellworld Party at an old mansion after receiving invites through the game. Mike, Derrick and Allison are enthusiastic about the party, while Chelsea reluctantly accompanies them. Jake, who is still very much distressed by Adam's death, only agrees to show up after a female Hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. The quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by Philip Lemarchand), and provides them with cell phones to communicate with other guests. As the party progresses, Allison, Derrick and Mike find themselves trapped in separate parts of the house, and are gruesomely killed by the Host, Pinhead, or Cenobite minions Chatterer II and Bound. Jake and Chelsea become mysteriously invisible to other party guests, and are stalked by the Host and the Cenobites. Holing herself up in the attic, Chelsea finds items belonging to Adam, and discovers that the host is his father, who blames his son's friends for not helping break his addiction. Chelsea and Jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. The Host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilty consciences. Before leaving, he lets Chelsea know that Allison, Derrick, and Mike have all perished in their respective caskets, and that only she and Jake remain alive. Chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from Chelsea's telephone. Looking towards the house, Chelsea sees Adam standing in the window. Later, the Host sits in a bedroom, going through a suitcase containing Adam's possessions. He finds and opens the actual Lament Configuration, which summons the real Cenobites. Pinhead praises Adam's ingenuity and mocks the Host's disbelief before Chatterer and Bound tear him to pieces. Jake and Chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the Host, who suddenly appears in the back seat. The two almost crash the car but are able to stop it. The last scene shows the police entering the bedroom in which the Host opened the box, the walls blood-smeared and the box lying on the floor.

Tags: good versus evil, revenge, neo noir, murder, violence

5.1 Getting a sense of the movie synopsis texts

In [23]:

```
#Utility functions for feature extraction
#Returns the count of 'http' elements present in a string. Return 0 otherwise.
def count_http(string):
    if string.__contains__("http"):
        return string.count("http")
    else:
        return int(0)

#Returns the number of times a reference link is present in a string
def count_href(string):
    if string.__contains__("a href"):
        return string.count("a href")
    else:
        return int(0)

#Number of times a greater than sign appears in a string
def count_greater(string):
    if string.__contains__(">"):
        return string.count(">")
    else:
        return int(0)
```

In [24]:

```
#Simple feature engineering
basic_feats = pd.DataFrame()
basic_feats["Length_Title"] = dataframe['title'].apply(lambda x: len(str(x))) #Length of RAW Title text
basic_feats["Length_Plot_Synopsis"] = dataframe['plot_synopsis'].apply(lambda x: len(str(x))) #Length of RAW body text
basic_feats['count_plot_synopsis_http'] = dataframe['plot_synopsis'].apply(lambda x: count_http(str(x))) #Lazy way to count the number of URLs present in a body text. Not 100% accurate, but close enough
basic_feats['count_plot_synopsis_href'] = dataframe['plot_synopsis'].apply(lambda x: count_href(str(x))) #Lazy way to count the reference to an external site. Not 100% accurate, but close enough
basic_feats['count_plot_synopsis_grtsign'] = dataframe['plot_synopsis'].apply(lambda x: count_greater(str(x))) #Very lazy way to count html tags present in a string. Not 100% accurate, but close enough

#Save the dataset containing basic features
basic_feats.to_csv('basic_feats.csv', columns=basic_feats.columns)
basic_feats.head(3)
```

Out[24]:

	Length_Title	Length_Plot_Synopsis	count_plot_synopsis_http	count_plot_synopsis_href	count_plot_synopsis_grtsign
0	1	3657	0	0	0
1	7	2057	0	0	0
2	3	4193	0	0	0

In [25]:

```
basic_feats['count_plot_synopsis_grtsign'].value_counts()
```

Out[25]:

```
0      14766
1         6
2         5
5         1
20        1
4         1
10        1
Name: count_plot_synopsis_grtsign, dtype: int64
```

5.2 High level statistics of the dataset containing simple features

In [26]:

```
#Get a high level stats of the given dataset
basic_feats.describe()
```

Out[26]:

	Length_Title	Length_Plot_Synopsis	count_plot_synopsis_http	count_plot_synopsis_href	count_plot_synopsis_grtsi
count	14781.000000	14781.000000	14781.000000	14781.0	14781.0000
mean	15.912658	5140.702118	0.000068	0.0	0.0037
std	8.502424	4945.551360	0.008225	0.0	0.1958
min	1.000000	442.000000	0.000000	0.0	0.0000
25%	10.000000	2491.000000	0.000000	0.0	0.0000
50%	14.000000	3825.000000	0.000000	0.0	0.0000
75%	20.000000	5760.000000	0.000000	0.0	0.0000
max	92.000000	63959.000000	1.000000	0.0	20.0000

In [27]:

```
#Get the percentage of movies which does not have a http reference included in their body
zero = basic_feats[basic_feats['count_plot_synopsis_http'] == 0].shape[0]
per = (zero/basic_feats.shape[0]) * 100
print("Percentage of movie plots which does not have any http reference URL in the body text: {:.2f}%".format(per))

#Get the percentage of questions which are provided with external reference links in their body text
zero = basic_feats[basic_feats['count_plot_synopsis_href'] == 0].shape[0]
per = (1-zero/basic_feats.shape[0]) * 100
print("Percentage of movie plots which does not have any external reference: {:.2f}%".format(per))
```

Percentage of movie plots which does not have any http reference URL in the body text: 99.99%
Percentage of movie plots which does not have any external reference: 0.00%

Observations from the above analysis:

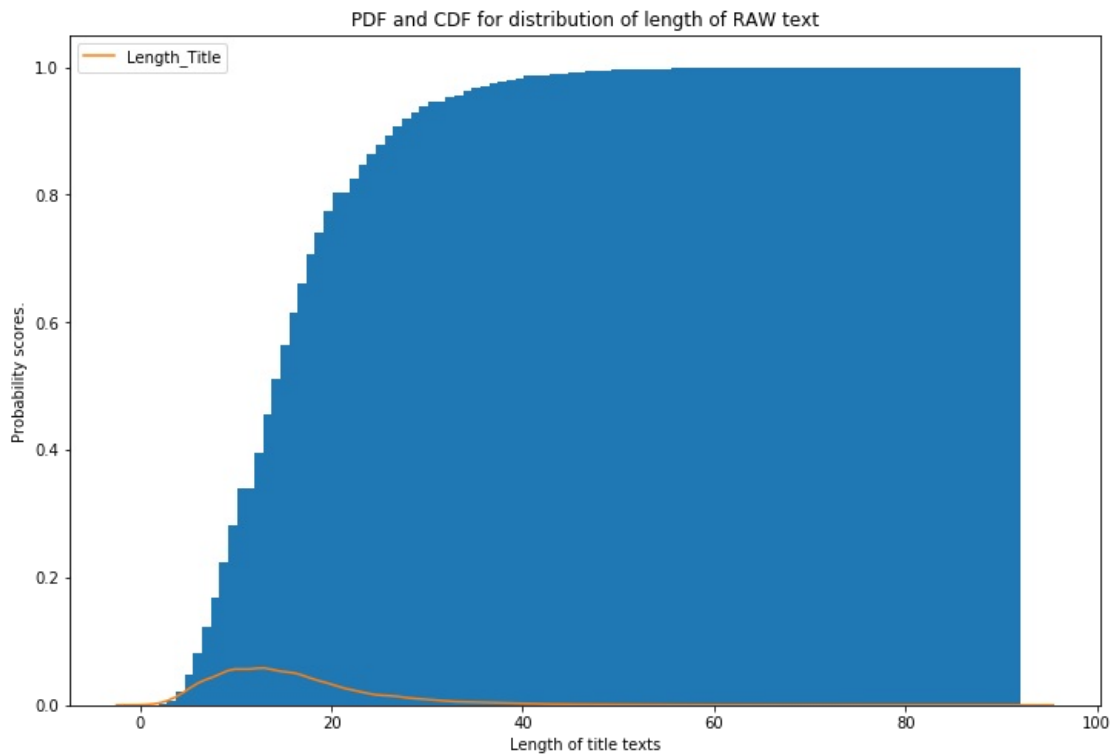
1. A quick high level statistic revealed that the average length of movie plots is somewhere around 5140.
2. The maximum length of plot synopsis is as high as 63959 characters and the minimum length is 442 characters!
3. The average length of movie title is somewhere around 15
4. 92 is the maximum string length for a movie title and 1 is the minimum string length for a movie title.
5. 99.99% movie plots does not have any http reference in their body.
6. There are 20 movie plots which has the 'greater than' sign.
7. Almost all the reviews has punctuation marks and contracted words in them.

3.1.8 Histograms of some of the extracted features

In [28]:

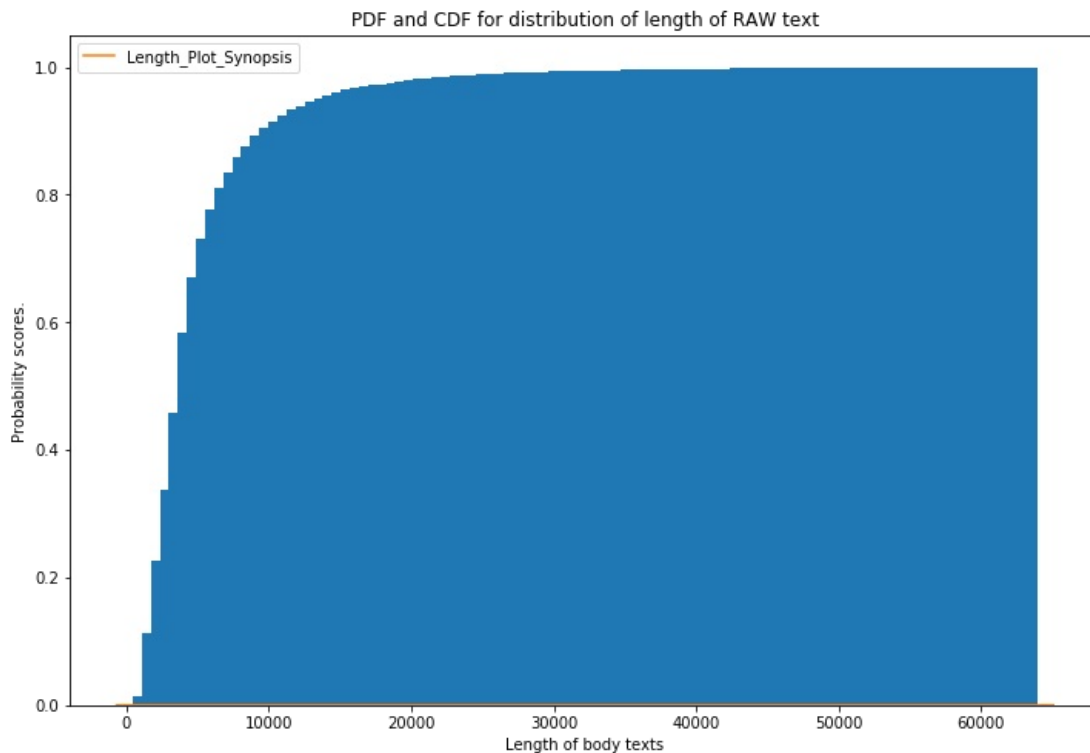
```
import scipy.stats as ss
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns # for nicer graphics

plt.figure(figsize=(12, 8))
myHist = plt.hist(basic_feats['Length_Title'].values, 100, density=True, cumulative=True)
plt.title('PDF and CDF for distribution of length of RAW text')
plt.xlabel('Length of title texts')
plt.ylabel('Probability scores.')
sns.kdeplot(basic_feats['Length_Title']);
plt.show()
```



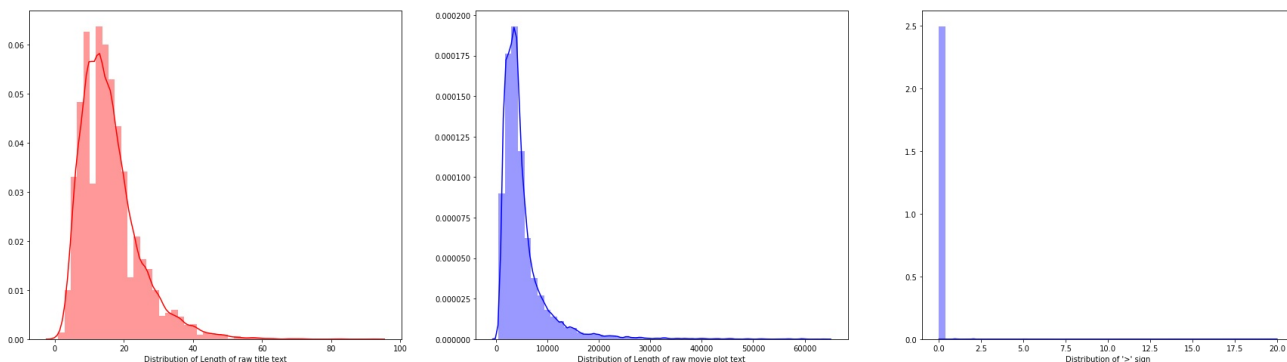
In [29]:

```
plt.figure(figsize=(12, 8))
myHist = plt.hist(basic_feats['Length_Plot_Synopsis'].values, 100, density=True, cumulative=True)
plt.title('PDF and CDF for distribution of length of RAW text')
plt.xlabel('Length of body texts')
plt.ylabel('Probability scores.')
sns.kdeplot(basic_feats['Length_Plot_Synopsis']);
plt.show()
```



In [30]:

```
#Draw only PDF
plt.figure(figsize=(30, 8))
plt.subplot(1,3,1)
sns.distplot([basic_feats['Length_Title']], color = 'red', axlabel="Distribution of Length of raw title text")
plt.subplot(1,3,2)
sns.distplot([basic_feats['Length_Plot_Synopsis']], color = 'blue', axlabel="Distribution of Length of raw movie plot text")
plt.subplot(1,3,3)
sns.distplot([basic_feats['count_plot_synopsis_grtsign']], color = 'blue', axlabel="Distribution of '>' sign")
plt.show()
```



Observations:

1. We can see most of the title texts has median length around 15.
2. The median length of the plot synopsis is around 5000. The distribution of movie plot length is highly skewed towards the left.
3. There are very few movies which has plot synopsis length greater than 20000 characters.

6. Analysis of Tags

6.1 Total number of unique tags

In [6]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'sp
lit()' will tokenize each tag using space.
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(tag_data['tags'])
```

In [42]:

```
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 : 14780
Number of unique tags present in the entire dataset: 71

In [43]:

```
#'get_feature_name()' gives us the vocabulary.
tags = vectorizer.get_feature_names()
print("Let's look at all the unique tags we have :\n\n", tags[:71])
```

Let's look at all the unique tags we have :

```
['absurd', 'action', 'adult comedy', 'allegory', 'alternate history', 'alternate reality', 'anti wa
r', 'atmospheric', 'autobiographical', 'avant garde', 'blaxploitation', 'bleak', 'boring', 'brainwas
hing', 'christian film', 'claustrophobic', 'clever', 'comedy', 'comic', 'cruelty', 'cult', 'cute', '
dark', 'depressing', 'dramatic', 'entertaining', 'fantasy', 'feel-good', 'flashback', 'good versus e
vil', 'gothic', 'grindhouse film', 'haunting', 'historical', 'historical fiction', 'home movie', 'ho
rror', 'humor', 'insanity', 'inspiring', 'intrigue', 'magical realism', 'melodrama', 'murder', 'myst
ery', 'neo noir', 'non fiction', 'paranormal', 'philosophical', 'plot twist', 'pornographic', 'prank
', 'psychedelic', 'psychological', 'queer', 'realism', 'revenge', 'romantic', 'sadist', 'satire', 's
ci-fi', 'sentimental', 'storytelling', 'stupid', 'suicidal', 'suspenseful', 'thought-provoking', 'tr
agedy', 'violence', 'western', 'whimsical']
```

6.2 Number of times a tag appeared

In [44]:

```
#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 #axis=0 for columns. Column contain the number of times the tags have occurred
result = dict(zip(tags, freqs))
```

In [46]:

```
#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(10)
```

Out[46]:

	Tags	Counts
0	absurd	270
1	action	660
2	adult comedy	128
3	allegory	138
4	alternate history	102
5	alternate reality	205
6	anti war	118
7	atmospheric	396
8	autobiographical	44
9	avant garde	220

6.3 Tags which are present the most number of times

In [47]:

```
#Sort the tags according to their number of occurences.  
#We see that murder, violence, flashback, romantic, cult are the 5 most frequently occuring tags.  
#We will visualize this distribtuion in a graph  
tag_df_sorted = tag_df.sort_values(['Counts'], ascending=False)  
tag_counts = tag_df_sorted['Counts'].values  
tag_df_sorted.head(10)
```

Out[47]:

	Tags	Counts
43	murder	5771
68	violence	4423
28	flashback	2937
57	romantic	2900
20	cult	2647
56	revenge	2465
52	psychedelic	1897
17	comedy	1858
65	suspenseful	1086
29	good versus evil	875

In [48]:

```
#tag_counts contains how many times each tags appeared in the entire dataset  
tag_counts
```

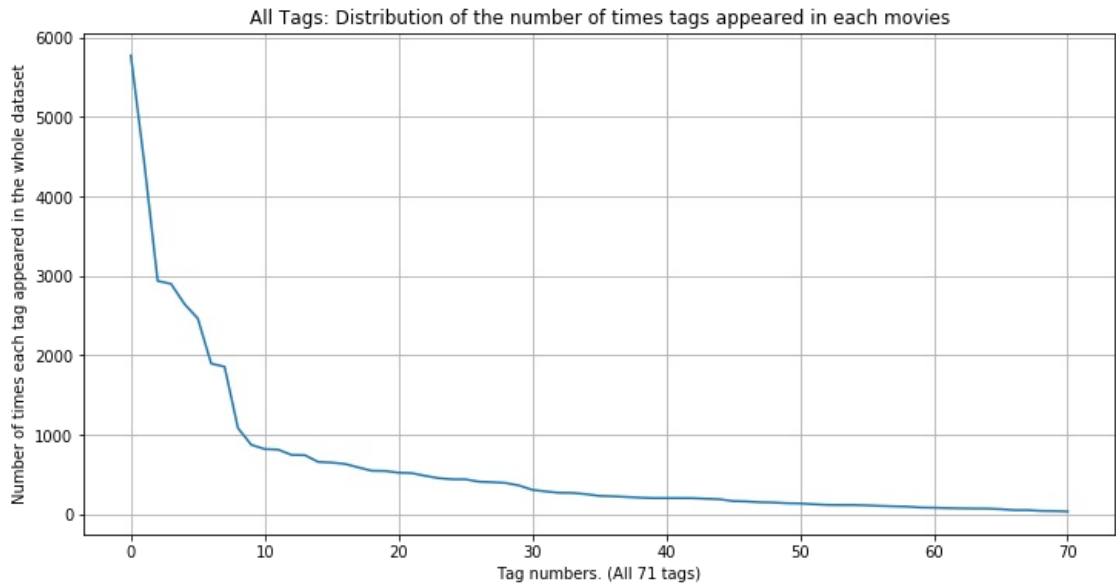
Out[48]:

```
array([5771, 4423, 2937, 2900, 2647, 2465, 1897, 1858, 1086, 875, 822,  
      815, 749, 745, 660, 652, 635, 591, 549, 546, 525, 519,  
      485, 456, 442, 441, 412, 405, 396, 364, 309, 289, 272,  
      270, 255, 233, 228, 220, 211, 205, 205, 205, 204, 197,  
      190, 168, 163, 153, 150, 141, 138, 128, 120, 118, 118,  
      114, 107, 102, 98, 87, 84, 79, 76, 74, 73, 66,  
      54, 54, 44, 42, 37])
```

6.4 Analysis of Tags : Distribution of all tags, i.e the number of times each tag appeared in movie synopses.

In [49]:

```
#Get the distribution information
plt.figure(figsize=(12, 6))
plt.plot(tag_counts)
plt.title("All Tags: Distribution of the number of times tags appeared in each movies")
plt.grid()
plt.xlabel("Tag numbers. (All 71 tags)")
plt.ylabel("Number of times each tag appeared in the whole dataset")
plt.show()
```



In [50]:

```
#Get a high level statistical view of the tags data
tag_df_sorted.describe()
```

Out[50]:

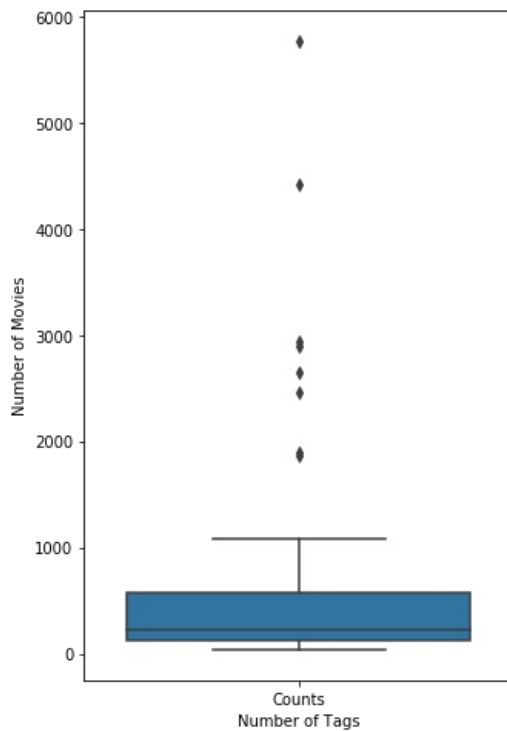
	Counts
count	71.000000
mean	621.816901
std	1016.487149
min	37.000000
25%	119.000000
50%	233.000000
75%	570.000000
max	5771.000000

In [22]:

```
#Using boxplot plot to get a sense of the quantile values
plt.figure(figsize=(5, 8))
sns.boxplot(data = tag_df_sorted)
plt.xlabel("Number of Tags")
plt.ylabel("Number of Movies")
```

Out[22]:

Text(0, 0.5, 'Number of Movies')



Key Observations:

1. 75% of tags occurs less than 570 times across different movies.
2. 25% of tags occurs less than 119 times across different movies.
3. The maximum number of times a tag occurs in a movie is 5771

In [35]:

```
#Store tags greater than 1K in one list
list_tags_grt_thn_1k = tag_df_sorted[tag_df_sorted.Counts>1000].Tags
#Print the length of the list
print ('{} Tags are used more than 1000 times'.format(len(list_tags_grt_thn_1k)))

# Store tags greater than 5K in one list
list_tags_grt_thn_5k = tag_df_sorted[tag_df_sorted.Counts>5000].Tags
#Print the length of the list.
print ('{} Tags are used more than 5000 times'.format(len(list_tags_grt_thn_5k)))

#Tags with the most frequency
print("Most frequently occurring tag: {}".format(tag_df_sorted.iloc[0][0]))
print("Number of times {} occurs: {}".format(tag_df_sorted.iloc[0][0],tag_counts[0]))
```

9 Tags are used more than 1000 times
1 Tags are used more than 5000 times
Most frequently occurring tag: murder
Number of times murder occurs: 5771

Observations:

1. There are total 9 tags which are used more than 1000 times.
2. 1 tags are used more than 5000 times.
3. Most frequent tag (i.e. 'murder') is used 5771 times.
4. Since some tags occur much more frequently than others, Micro-averaged F1-score is the appropriate metric for this problem.

6.5 Tags Per Question

In [51]:

```
#Storing the count of tag in each question in list 'tag_count'
tag_quest_count = tag_dtm.sum(axis=1).tolist()

#Converting each value in the 'tag_quest_count' to integer.
tag_quest_count=[int(j) for i in tag_quest_count for j in i]
print ('We have total {} datapoints.'.format(len(tag_quest_count)))

print(tag_quest_count[:50])
```

We have total 14780 datapoints.

[1, 4, 2, 6, 1, 1, 2, 3, 1, 5, 1, 13, 1, 2, 1, 1, 15, 3, 1, 1, 2, 2, 2, 1, 5, 1, 1, 6, 11, 2, 3, 2, 4, 1, 4, 6, 3, 1, 5, 1, 5, 1, 1, 7, 7, 3, 1, 4, 4, 2]

In [52]:

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

Maximum number of tags per question: 25

Minimum number of tags per question: 1

Avg. number of tags per question: 2.987077

In [75]:

```
#How many movies have tags less than or equal to 3?
tag_greater_than_avg_count = list(filter(lambda x: x<=3, tag_quest_count))
len(tag_greater_than_avg_count)
```

Out[75]:

10551

In [77]:

```
#How many movies have tags less than or equal to 4?
tag_greater_than_avg_count = list(filter(lambda x: x<=4, tag_quest_count))
len(tag_greater_than_avg_count)
```

Out[77]:

11789

In [78]:

```
#How many movies have tags less than or equal to 5?
tag_greater_than_avg_count = list(filter(lambda x: x<=5, tag_quest_count))
len(tag_greater_than_avg_count)
```

Out[78]:

12705

In [79]:

```
#How many movies have tags less than or equal to 6?
tag_greater_than_avg_count = list(filter(lambda x: x<=6, tag_quest_count))
len(tag_greater_than_avg_count)
```

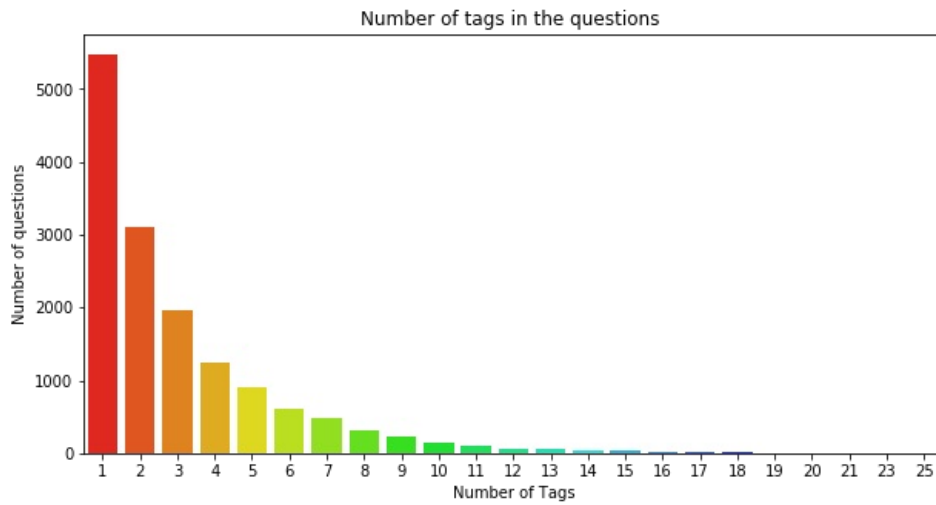
Out[79]:

13311

6.6 Histogram for distribution of tags.

In [38]:

```
plt.figure(figsize=(10,5))
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the movies ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of movies")
plt.show()
```



Observations from the above analysis.

1. Maximum number of tags per movies: 25
2. Minimum number of tags per movies: 1
3. Avg. number of tags per question: 2.987077
4. Most of the movie plots has tags between 1 and 6. There are lesser number of movie synopses which has tags 7, 8, 9, 10, 11.
5. There are even lesser number of movies which has tags greater than 12, with the maximum number of tags going to as high as 25.

6.6 Word Cloud for the most frequently occurring tags in the movie synopses plots

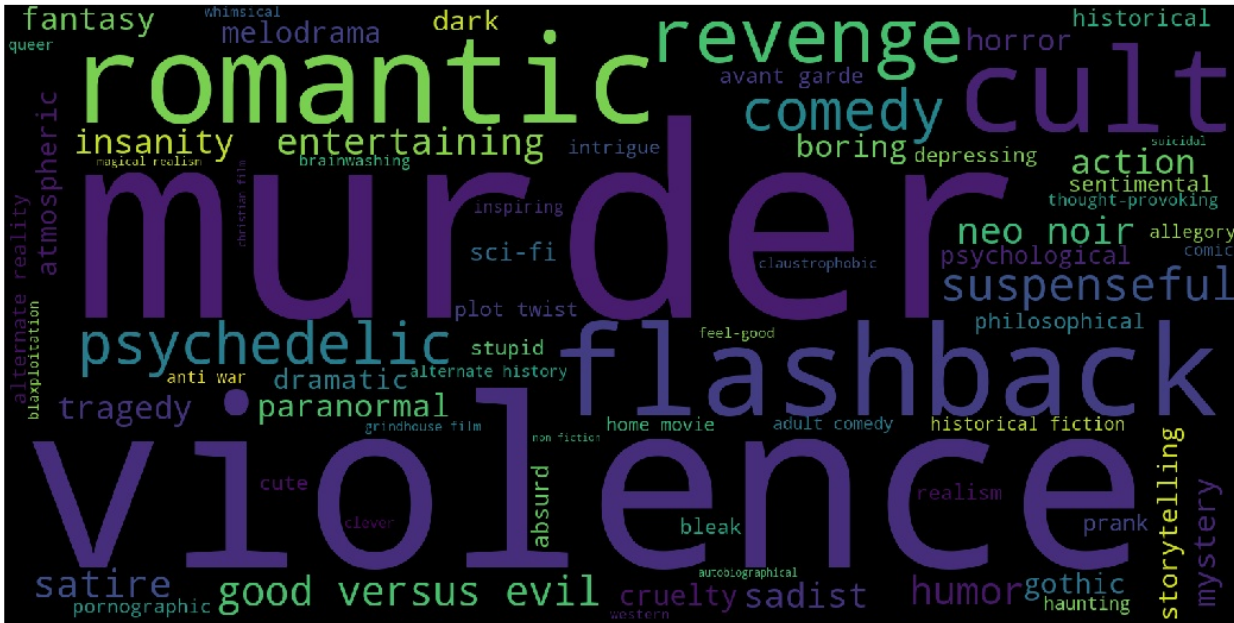
In [45]:

```
# Plotting 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=(15,10))
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 : 0:00:02.301621

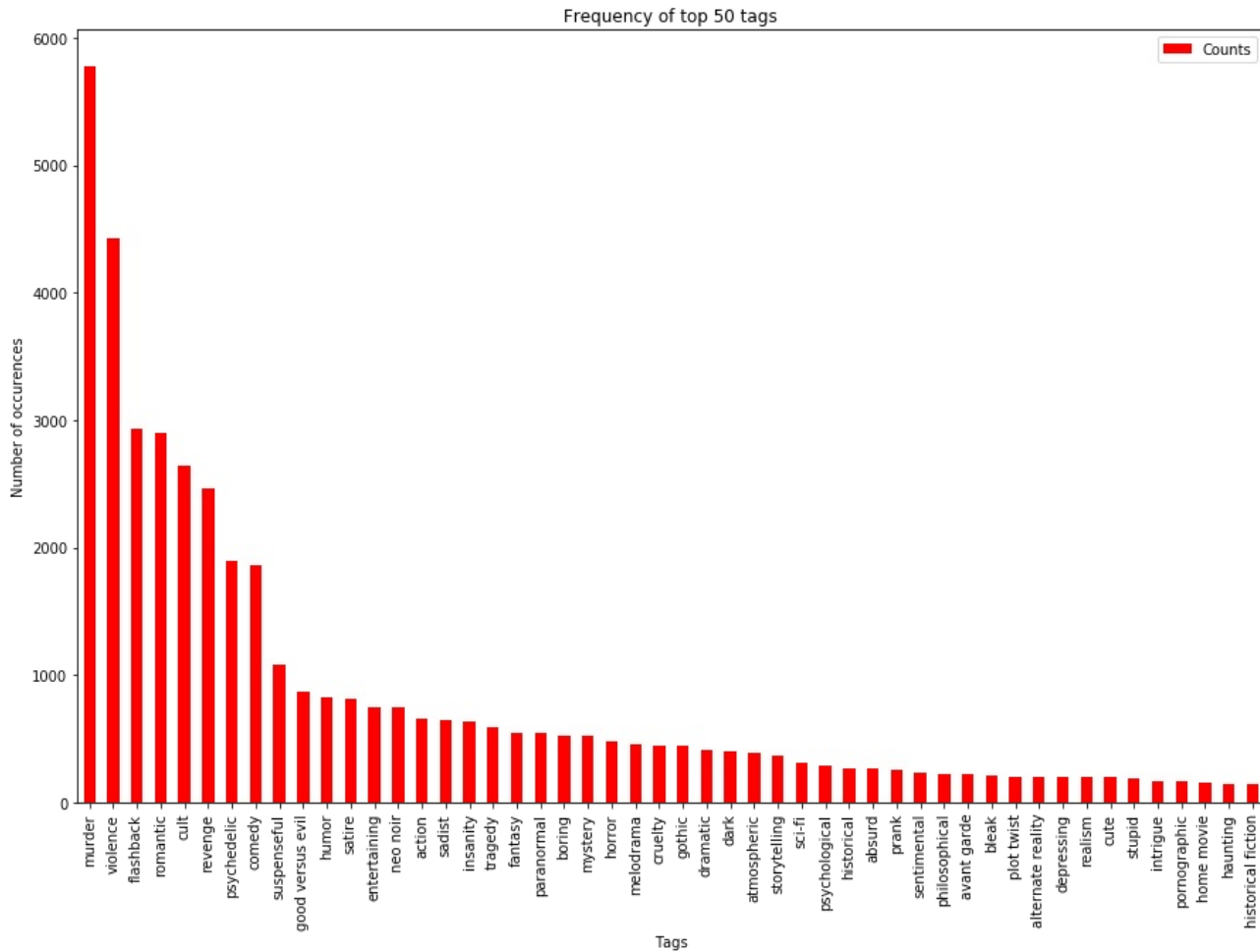
Observations from the above word cloud.

1. A look at the word cloud shows that "murder", "violence", "flashback", "romantic", "cult" are the most frequently occurring tags in the movie synopsis plots.
2. There are lots of tags which occurs less frequently like "comedy", "psychedelic", "horror", "entertaining", "humor" etc.

6.8 Distribution of frequently occurring tags by their frequency

In [39]:

```
i=np.arange(50)
tag_df_sorted.head(50).plot(kind='bar', figsize=(15,10), rot=90, color='red')
plt.title('Frequency of top 50 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Number of occurrences')
plt.show()
```



Observations from the above plot.

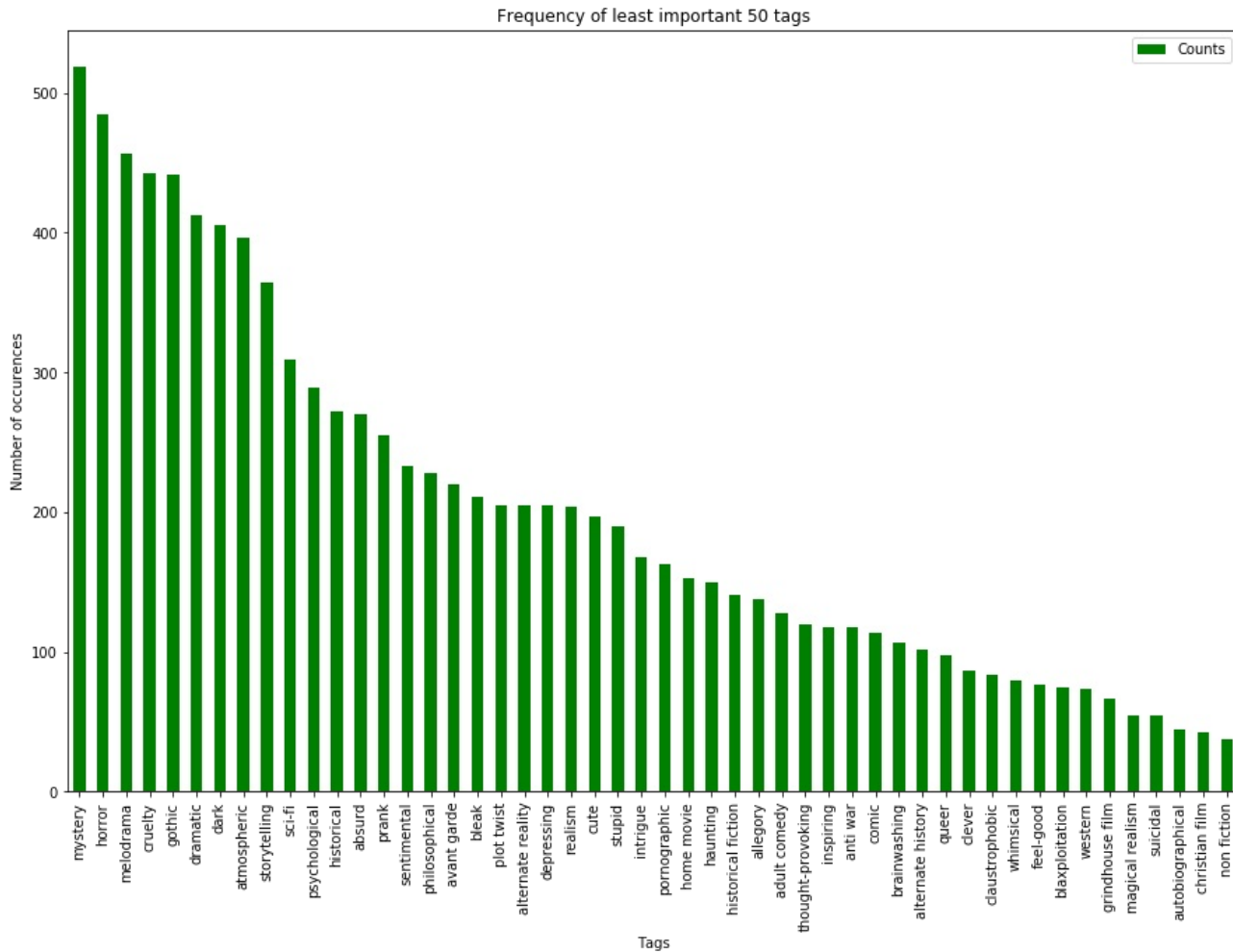
1. "Murder" is the most frequently occurring tags followed by "violence", "flashback", "romantic"
2. "Murder" and "Violence" occurs in more than 5000 movies.
3. "Flashback", "Romantic", "Cult", "Revenge" tags occurs in more than 2000 movies.
4. Almost all other remaining tags occurs less than 1000 number of times across the entire dataset.

The above analysis is done on the entire dataset, later we will split the data into train and test data. We will use the train data and validation data for training and cross validating our model. We will use the test data for evaluating our models performance on unseen data.

6.8 The least frequently occurring tags

In [40]:

```
i=np.arange(50)
tag_df_sorted.tail(50).plot(kind='bar', figsize=(15,10), rot=90, color='green')
plt.title('Frequency of least important 50 tags')
plt.xticks(i, tag_df_sorted['Tags'][-50:])
plt.xlabel('Tags')
plt.ylabel('Number of occurrences')
plt.show()
```



In [41]:

```
#These are the least frequently occurring tags
print("The tags which occurs frequently across the entire dataset are: \n")
print(list(tag_df_sorted['Tags'][-50:]))
```

The tags which occurs least frequently across the entire dataset are:

```
['mystery', 'horror', 'melodrama', 'cruelty', 'gothic', 'dramatic', 'dark', 'atmospheric', 'storytelling', 'sci-fi', 'psychological', 'historical', 'absurd', 'prank', 'sentimental', 'philosophical', 'avant garde', 'bleak', 'plot twist', 'alternate reality', 'depressing', 'realism', 'cute', 'stupid', 'intrigue', 'pornographic', 'home movie', 'haunting', 'historical fiction', 'allegory', 'adult comedy', 'thought-provoking', 'inspiring', 'anti war', 'comic', 'brainwashing', 'alternate history', 'queer', 'clever', 'claustrophobic', 'whimsical', 'feel-good', 'blaxploitation', 'western', 'grindhouse film', 'magical realism', 'suicidal', 'autobiographical', 'christian film', 'non fiction']
```

6.9 EDA using K-Means Clustering on BOW representations of tags

In [49]:

```
from sklearn.cluster import KMeans

#Elbow method to determine the best value of K in K-Means clustering.
def plot_elbow(sumOfSquaredErrors, n_clusters, vectorizationType):
    '''This function is used to plot the elbow curve for sum of squared errors vs cluster values and obtain the optimal value of the hyperparameter K.'''

    k_values = n_clusters
    loss = sumOfSquaredErrors

    #Plot K Values vs Loss Values
    plt.figure(figsize=(35,8))
```

```

plt.plot(k_values, loss, color='red', linestyle='dashed', linewidth=5, marker='o', markerfacecolor='blue', markersize=10)
for xy in zip(k_values, np.round(loss, 3)):
    plt.annotate('%s, %s' % xy, xy=xy, textcoords='data')
plt.title('K vs Loss for {} model'.format(vectorizationType))
plt.xlabel('Number of clusters')
plt.ylabel('Loss (Sum of Squared Errors)')
plt.show()

optimal_k = input("Please select the optimal number of clusters from the above elbow plot and press enter : ")
print("The optimal number of clusters selected from the elbow method is {}".format(optimal_k))

return optimal_k

#Function to perform KMeans Clustering.
def KMeansPlusPlus(tags_vectors):
    '''This function is used for multiple method calls which would determine the optimal value of k. The loss is calculated for each clusters and the value of the optimal number of clusters is obtained by visually examining the elbow plot. At the end the k-means algorithm will be run with the best value of K selected from the elbow plot'''
    t_start = datetime.now()
    sumOfSquaredErrors = []
    n_clusters = range(1, 25)
    k_means = [KMeans(n_clusters=i, n_init=5, init='k-means++', n_jobs=8, random_state=0) for i in n_clusters] # algorithm = elkan for dense data data, default: algorithm = auto
    k_means_centroids = [k_mean.fit(tags_vectors) for k_mean in k_means]
    sumOfSquaredErrors = [k_mean.inertia_ for k_mean in k_means_centroids] # Inertia: Sum of distances of samples to their closest cluster center
    optimal_k = int(plot_elbow(sumOfSquaredErrors, n_clusters, "BOW"))

    #Run k-medoids with the optimal number of clusters obtained from the elbow method
    kmeans = KMeans(n_clusters=optimal_k, init='k-means++', algorithm='auto', n_jobs=8, random_state=0).fit(tags_vectors)
    print("Time taken to perform K-Means clustering on Tags data: ", datetime.now() - t_start)

    return kmeans, optimal_k

#Function to draw word clouds for each clusters.
from wordcloud import WordCloud
def word_clouds(kmeans_object, tags_corpus):
    #Labels of each data point
    labels=kmeans_object.labels_
    clusters_dict = {i: np.where(labels == i)[0] for i in range(optimal_k)}
    # Transform this dictionary into list (if you need a list as result)
    clusters_list = []
    print("The number of datapoints in each cluster are as follows : ")
    for key, value in clusters_dict.items():
        temp = [key, value]
        clusters_list.append(temp)
        print("Cluster = {}, Number of data points = {}".format(key+1, len(value)))

    from wordcloud import WordCloud
    for cluster_number in range(optimal_k-2):
        cluster = [clusters_dict[cluster_number][i] for i in range(clusters_dict[cluster_number].size)]

        reviews_cluster = []
        for i in cluster:
            reviews_cluster.append(tags_corpus[i])

        review_corpus = ""
        for review in reviews_cluster:
            review_corpus = review_corpus + " " + review

        # lower max_font_size
        wordcloud = WordCloud(width=800, height=450, margin=2, prefer_horizontal=0.9, scale=1, max_words=75, min_font_size=4, random_state=42, background_color='black', contour_color='black', repeat=False).generate(str(review_corpus))

        plt.figure(figsize=(16,9))
        plt.title("Word Cloud for Cluster {}".format(cluster_number+1))
        plt.imshow(wordcloud, interpolation="bilinear")
        plt.axis("off")
        plt.show()

```

In [42]:

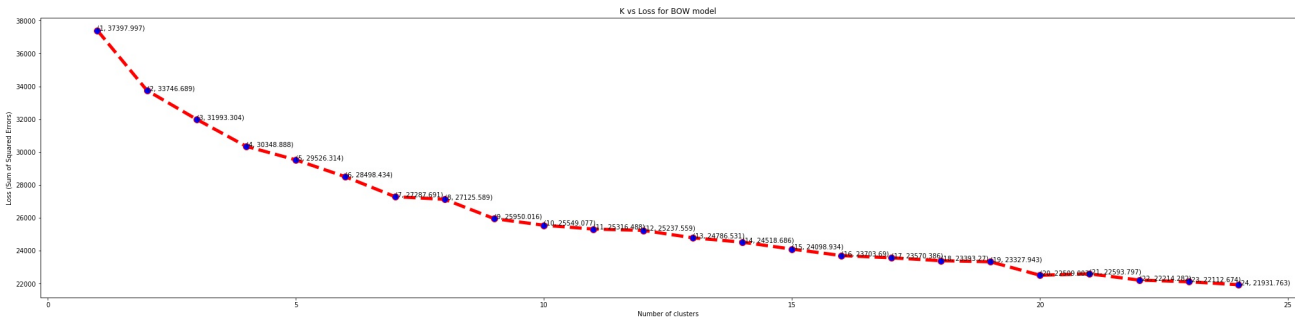
```
tag_data.head()
```

Out[42]:

	tags
1	flashback
2	suspenseful, neo noir, murder, violence
3	cult, violence
4	murder, anti war, violence, flashback, tragedy...
5	murder

In [100]:

```
#Taking all the tags
tags_corpus=tag_data['tags'].apply(lambda x: str(x)) #Avoid encoding problems
cv_object = CountVectorizer(tokenizer = tokenize).fit(tags_corpus) #Initializing the BOW constructor
tags_vectors = cv_object.transform(tags_corpus) #Creating BOW vectors of all the tags
kmeans_object, optimal_k = KMeansPlusPlus(tags_vectors) #KMeans++ Algorithm function call to get the best kmeans
object and optimal number of clusters
```



Please select the optimal numberof clusters from the above elbow plot and press enter : 7
The optimal number of clusters selected from the elbow method is 7
Time taken to perform K-Means clustering on Tags data: 0:04:52.614027

In [107]:

```
#Plot word clouds of similar tags
word_clouds(kmeans_object, tags_corpus)
```

The number of datapoints in each cluster are as follows :

- Cluster = 1, Number of data points = 5060
- Cluster = 2, Number of data points = 2114
- Cluster = 3, Number of data points = 1194
- Cluster = 4, Number of data points = 1756
- Cluster = 5, Number of data points = 1049
- Cluster = 6, Number of data points = 1787
- Cluster = 7, Number of data points = 1820

[illegible]

[illegible][illegible]

1. In all the 5 clusters we see that the tags "violence","murder" has a tendency to occur together.
2. We can also see tags like "revenge" which has a tendency to occur with tags like both "murder" and "romantic".
3. Tags like "cult", "evil", "violence" has a higher chance of occurring together.
4. Tags like "comedy","melodrama" and "entertaining" has a higher chance of occurring together.
5. "Psychedelic", "Suspenseful" and "boring" has higher chances of occurring together.

1. We will remove the html tags (if any) from the movie plots
2. Remove any punctuations or limited set of special characters like , or . or # etc.
3. Check if the word is made up of english letters and is not alpha-numeric
4. 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)
5. Convert the word to lowercase
6. Remove Stopwords
7. Perform decontraction of words
8. Finally Snowball Stemming the word (it was observed to be better than Porter Stemming)

```
#Load the de-duplicated dataset
start = datetime.now()
con = sqlite3.connect('train_no_dup.db')
dataframe = pd.read_sql_query("""SELECT * FROM no_dup_train""", con)
con.close()

dataframe.head()
```

index		title		plot_synopsis		tags	split
0	0	\$	Set in Hamburg, West Germany, several criminal...			murder	test
1	1	\$windle	A 6th grader named Griffin Bing decides to gat...			flashback	train
2	2	'71	Gary Hook, a new recruit to the British Army, ...		suspenseful, neo noir, murder, violence		train
3	3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...			cult, violence	train
4	4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...			train

Print the original movie plots

In [51]:

```
#Printing some random movie plots from the deduplicated dataset.
sent_1 = dataframe['plot_synopsis'].values[0]
print(sent_1)
print("\nTags: {}".format(dataframe['tags'].values[0]))
print("="*215)

sent_2 = dataframe['plot_synopsis'].values[1000]
print(sent_2)
print("\nTags: {}".format(dataframe['tags'].values[1000]))
print("="*215)

sent_3 = dataframe['plot_synopsis'].values[1500]
print(sent_3)
print("\nTags: {}".format(dataframe['tags'].values[1500]))
print("="*215)

sent_4 = dataframe['plot_synopsis'].values[4900]
print(sent_4)
print("\nTags: {}".format(dataframe['tags'].values[4900]))
print("="*215)
```

Set in Hamburg, West Germany, several criminals take advantage of the German privacy bank laws to use safe deposit boxes in a German bank to store large amounts of illicit cash. These include a Las Vegas mobster known only as the Attorney (Robert Webber) as well as a ruthless drug smuggler known as the Candy Man (Arthur Brauss) and a crooked overbearing U.S. Army sergeant (Scott Brady) and his meek-mannered partner the Major (Robert Stiles), who conspire on a big heroin and LSD smuggling score. Joe Collins (Warren Beatty), an American bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than \$1.5 million) with the help of Dawn Divine (Goldie Hawn), a hooker with a heart of gold. On the day of the robbery, Joe has Dawn phone in a bomb threat to the bank president, Mr. Kessel (Gert Fröbe), to create a diversion. Joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. The bank is closed and evacuated while Joe uses duplicate keys to empty the criminals' three safe deposit boxes into Dawn's large-size deposit box. (It is implied that Joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with Dawn.) Despite the fact that Kessel insists on burning through the wall to rescue Joe instead of waiting for the time lock to open, Joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. The next day, the three criminals, one by one, discover that their boxes are empty and they cannot complete their schemes or go to the police to report the thief. The Attorney flees the country while the others (Sarge, his partner the Major, and the Candy Man) search Dawn Divine's apartment as she was their common link and find clues that connect her to Joe. Sarge calls Kessel to get Joe's home address, but Joe is quickly tipped off by Kessel and he hurriedly sends Dawn to the train station with a suitcase packed with her take (\$765,000) promising to meet her later someplace out of the country. A long climactic chase begins as Dawn gives the Major the slip at the train station while the Candy Man and the Sarge chase Joe across a rail yard and through the Elbe Tunnel. Joe escapes on a car carrier truck, lugging his suitcase, but the Candy Man and the Sarge follow and catch up in the morning at a frozen lake in the countryside, where the Candy Man crashes a car through the ice and drowns while attempting to run Joe down with a stolen car. Joe escapes again by hopping a train, but during the night the Sarge catches up to him only to find that Joe's suitcase contains nothing but a bottle of champagne and wads of newspaper. They conclude that Dawn double-crossed Joe by repacking the suitcases while he was getting the car, and the Sarge proposes a plan to Joe to go after Dawn together. But, upon drinking a swallow of the champagne, the Sarge instantly goes into violent convulsions and falls down dead. The bottle was one of three that the Candy Man had filled with a solution of concentrated LSD to sneak them through customs earlier in the film. Joe then disembarks from the train and walks away, apparently betrayed by Dawn. An epilog shows Dawn in a sunny climate in the USA, joyfully driving a gleaming new yellow Corvette, and then later cuddling in bed with an unseen someone. The other suitcase is sitting near the bed, and Joe's bomber jacket hangs on the coat rack. Dawn smugly explains to the person she was certain the criminals wouldn't kill him and leave themselves with no way to get the money.

Tags: murder

Years ago, a mob boss named Lucio Malatesta (George Touliatos) pinned the murder of rival Sammy Carbone (Gino Marrocco) on another rival named Angelo Allieghieri (Anthony Quinn), which led to Sammy's son Gianni vowing revenge.

Frankie Delano (Sylvester Stallone) has spent his life safeguarding Angelo as well as Angelo's daughter, Jennifer Barrett (Madeleine Stowe), whose unsavory husband Kip Barrett (Harry Van Gorkum) has had their young son Rawley (Ezra Perlman) placed in a boarding school against Jennifer's wishes. Jennifer was raised by her adoptive parents Whitney Towers (John Gilbert) and Peggy Towers (Dawn Greenthalgh) and is not aware that Angelo is her father.

After Angelo is killed in a restaurant by a hit man named Bruno (Billy Gardell), Frankie introduces himself, tells Jennifer who he is and what he has been doing.

A neurotic mess, Jennifer can barely handle the news that Kip is a philanderer, let alone the revelation that she is a gangster's daughter. But a DVD prepared by Angelo in the case of just such an event convinces Jennifer that it's the truth.

Jennifer certainly doesn't want a full-time bodyguard, even Frankie. She ditches Kip and then falls for Italian romance novelist Marcello (Raoul Bova), who lectures at her book club. Frankie has suspicions about Marcello, but his job is to stay on the sidelines.

Frankie rescues Jennifer from a string of attacks. With many of Angelo's enemies, including Lucio Malatesta, terminated, Frankie allows her to visit Italy with Marcello. But it turns out that Marcello

is actually Gianni Carboni, who had Angelo killed. And now Gianni plans to kill Jennifer. It's up to Frankie to protect her one more time.

Tags: violence, comedy, murder, flashback

=====

Nick Conklin (Michael Douglas) is a skilled motorcyclist and a tough veteran New York City police officer facing possible criminal charges; Internal Affairs believes Nick was involved with his partner who was caught taking criminal money in a corruption scandal. Nick is divorced from his wife, who has custody of their two children. Nick also has financial difficulties due to alimony and child support as well as other concerns. Nick reports to a criminal investigation hearing being run by two officers from Internal Affairs, a conference that doesn't go well for him. They ask Nick about his involvement with several officers under investigation. When Nick refuses to squeal on his comrades, Internal Affairs threatens him, suggesting he's as corrupt as the others in the department. While having a drink at a local Italian restaurant/bar, Nick and his partner Charlie Vincent (Andy Garcia) observe two Japanese men having what appears to be a friendly lunch with some Italian gangsters. Nick is increasingly suspicious of the group until another Japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the Japanese. As the man turns to leave, one of the Japanese men at the table says, in Japanese, "The Oyabun [Godfather] will not stand for this." The leader of the Japanese group chimes in, "As always, such a troublesome child." The Japanese man finds these remarks insulting and he slashes the man's throat, stabs another in the chest, and then walks out. Nick and Charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills Nick in a nearby slaughterhouse. The suspect turns out to be a Yakuza gangster by the name of Sato (Yusaku Matsuda). The situation is further complicated when Nick's superior officer, Captain Oliver (John Spencer), tells him that Sato is to be extradited to Osaka and given to the police there. Nick is angry that Sato will not be tried for murder in the United States, but agrees to escort him to Japan. Nick's captain also has an ulterior motive for sending Nick overseas; he believes the excursion will keep Nick from causing more trouble and exacerbating the already biased Internal Affairs investigation of him. On the plane, Nick and Charlie talk about Nick's situation and how Nick's own expenses are beyond his means to pay them. At one point, while Charlie is out of his seat, Sato notices Nick cheating at solitaire and contemptibly chuckles to himself. Nick cruelly hits his prisoner in the mouth and lies about it when Charlie returns and asks what happened. When they arrive in Osaka, men identifying themselves as Japanese police immediately meet them on the plane, display a "transfer document" printed in Japanese and take Sato into their custody, leaving the plane by the rear exit. As Nick and Charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in English, indicating that the first "cops" were impostors. Nick and Charlie are taken to the headquarters of the Osaka Prefecture of Police and questioned. They are also blamed for Sato's escape. After much haranguing by Nick (who shows xenophobia) towards the Japanese, who rarely acknowledge that they can speak English, he and Charlie are allowed to observe the hunt for Sato. However, the senior police officer emphasizes that they have no authority in Japan and it is illegal for them to carry their guns, which are confiscated. They are assigned to Masahiro Matsumoto (Takakura), a mild-mannered and experienced officer, who will be their guide. Throughout the investigation Nick behaves rudely, offending Matsumoto, while Charlie tries to be more polite. Taken to a murder scene at a local nightclub, Nick recognizes the murder victim as one of the men at the airport who took Sato into custody. While the dead man is examined by forensic experts, one of them removes a \$100 bill from his mouth. Nick makes contact with an American blond nightclub hostess, Joyce (Kate Capshaw), who explains that the Japanese public, including the giggling hostesses in the club, all believe that Nick and Charlie are not to be taken seriously because they allowed Sato to easily escape from custody, and represent American inefficiency and stupidity. Through Joyce, Nick discovers that Sato is fighting a gang war with a notorious crime boss, Sugai (Tom Isaburo Wakayama). Sato used to be a lieutenant for Sugai and now wants his own territory to rule. Sato had traveled to New York to disrupt a meeting with American Italian gangsters about a scheme being set up by Sugai involving the package Sato had taken in the restaurant. Having joined a police raid of a gang hideout without permission, Nick takes some \$100 bills from a table, which he later shows are forgeries by burning one. The next day Matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to America; Nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in Matsumoto's superior's office. He suggests that the package stolen by Sato was either more samples of the forged bills or plates to make more. Late one night, after spending a few hours in a nightclub with Matsumoto, Nick and Charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from Matsumoto. They are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals Charlie's raincoat and lures Charlie into an underground parking garage. Nick follows, shouting for Charlie to come back, but is separated from his partner by a security gate. The unarmed Nick then watches in horror as Sato and several of his bszoku gang members briefly torture Charlie using swords and knives, before Sato beheads him. Distraught, Nick is comforted by Joyce at her apartment. Matsumoto arrives with Charlie's belongings, including his NYPD badge, which Nick gives to Matsumoto, and Charlie's service pistol, which Nick keeps for himself. Matsumoto and Nick trail one of Sato's operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and Nick admits to Matsumoto that he had taken some money in New York, where he says there is no "black-and-white" procedure, only "gray" areas. Matsumoto disagrees, saying "theft is theft" and that Nick's illegal action disgraces all police, including Matsumoto and Charlie. Nick realizes Matsumoto is right and humbly accepts Matsumoto's advice. In the morning the woman retrieves from a bank strongbox a sample counterfeit note (printed only on one side) which she passes to one of Sato's gang on the street. Nick and Matsumoto tail the man to a steel foundry where they find Sato meeting with Sugai, and discover that the package that Sato had stolen in New York contains one of the printing plates for the American \$100 bill. Nick intervenes when Sato leaves the meeting and a gunfight ensues. Sato escapes again when Nick is arrested by the swarming police for using a gun in public, and told he will be sent back to New York in disgrace. Nick boards the plane for New York but is able to sneak off to pursue Sato on his own. He finds that Matsumoto has been suspended and demoted by his police force, a deep humiliation. Joyce helps him meet Sugai, who explains that making counterfeit U.S. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of Hiroshima and the loss of dignity he and his family faced in the aftermath of W

orld War II. Nick suggests a deal where Sugai can use Nick as an insignificant American to retrieve the stolen plate from Sato, leaving Sugai's reputation and hands clean.Sugai drops Nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. Nick is supplied with a shotgun. Sato arrives a short time later, as does Matsumoto. Matsumoto and Nick discover that Sato's men are planning a massacre. At the meeting table, Sato surrenders his single plate and requests recognition and his own territory. However, Sugai demands that Sato first atone for his offenses against the Yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. As he takes his position next to Sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between Sugai's and Sato's men. Sato escapes the fight on a dirt bike with Nick close behind. Nick is able to spill Sato off his bike and the two fight briefly, until Nick gains the advantage. The scene ends with Nick having to decide whether or not to kill Sato for Charlie and for all the humiliation he has suffered.The film ends with Matsumoto and Nick walking a handcuffed Sato into police HQ to the amazement of everyone and later receiving commendations, which Nick graciously accepts. At the airport, Nick thanks Matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. Underneath the shirt, Matsumoto finds the two counterfeit printing plates.

Tags: boring, neo noir, murder, violence, cult, romantic, suspenseful

The film introduces a circle of youths who are addicted to playing Hellworld, an online computer game based on the Hellraiser series. The film opens at the funeral of Adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. The remaining five friends blame themselves for not having prevented Adam's suicide. Two years later, they attend a private Hellworld Party at an old mansion after receiving invites through the game. Mike, Derrick and Allison are enthusiastic about the party, while Chelsea reluctantly accompanies them. Jake, who is still very much distressed by Adam's death, only agrees to show up after a female Hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. The quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by Philip Lemarchand), and provides them with cell phones to communicate with other guests. As the party progresses, Allison, Derrick and Mike find themselves trapped in separate parts of the house, and are gruesomely killed by the Host, Pinhead, or Cenobite minions Chatterer II and Bound. Jake and Chelsea become mysteriously invisible to other party guests, and are stalked by the Host and the Cenobites. Holing herself up in the attic, Chelsea finds items belonging to Adam, and discovers that the host is his father, who blames his son's friends for not helping break his addiction. Chelsea and Jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. The Host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilty consciences. Before leaving, he lets Chelsea know that Allison, Derrick, and Mike have all perished in their respective caskets, and that only she and Jake remain alive. Chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from Chelsea's telephone. Looking towards the house, Chelsea sees Adam standing in the window. Later, the Host sits in a bedroom, going through a suitcase containing Adam's possessions. He finds and opens the actual Lament Configuration, which summons the real Cenobites. Pinhead praises Adam's ingenuity and mocks the Host's disbelief before Chatterer and Bound tear him to pieces. Jake and Chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the Host, who suddenly appears in the back seat. The two almost crash the car but are able to stop it. The last scene shows the police entering the bedroom in which the Host opened the box, the walls blood-smeared and the box lying on the floor.

Tags: good versus evil, revenge, neo noir, murder, violence

Print the movie plots after removing URLs (if any)

In [52]:

```
#Remove urls from text python: https://stackoverflow.com/a/40823105/4084039
sent_1 = re.sub(r"http\S+", " ", sent_1)
sent_2 = re.sub(r"http\S+", " ", sent_2)
sent_3 = re.sub(r"http\S+", " ", sent_3)
sent_4 = re.sub(r"http\S+", " ", sent_4)

print(sent_1 + "\n")
print(sent_2 + "\n")
print(sent_3 + "\n")
print(sent_4 + "\n")
```

Set in Hamburg, West Germany, several criminals take advantage of the German privacy bank laws to use safe deposit boxes in a German bank to store large amounts of illicit cash. These include a Las Vegas mobster known only as the Attorney (Robert Webber) as well as a ruthless drug smuggler known as the Candy Man (Arthur Brauss) and a crooked overbearing U.S. Army sergeant (Scott Brady) and his meek-mannered partner the Major (Robert Stiles), who conspire on a big heroin and LSD smuggling score. Joe Collins (Warren Beatty), an American bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than \$1.5 million) with th

the help of Dawn Divine (Goldie Hawn), a hooker with a heart of gold. On the day of the robbery, Joe has Dawn phone in a bomb threat to the bank president, Mr. Kessel (Gert Fröbe), to create a diversion. Joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. The bank is closed and evacuated while Joe uses duplicate keys to empty the criminals' three safe deposit boxes into Dawn's large-size deposit box. (It is implied that Joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with Dawn.) Despite the fact that Kessel insists on burning through the wall to rescue Joe instead of waiting for the time lock to open, Joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. The next day, the three criminals, one by one, discover that their boxes are empty and they cannot complete their schemes or go to the police to report the thief. The Attorney flees the country while the others (Sarge, his partner the Major, and the Candy Man) search Dawn Divine's apartment as she was their common link and find clues that connect her to Joe. Sarge calls Kessel to get Joe's home address, but Joe is quickly tipped off by Kessel and he hurriedly sends Dawn to the train station with a suitcase packed with her take (\$765,000) promising to meet her later someplace out of the country. A long climatic chase begins as Dawn gives the Major the slip at the train station while the Candy Man and the Sarge chase Joe across a rail yard and through the Elbe Tunnel. Joe escapes on a car carrier truck, lugging his suitcase, but the Candy Man and the Sarge follow and catch up in the morning at a frozen lake in the countryside, where the Candy Man crashes a car through the ice and drowns while attempting to run Joe down with a stolen car. Joe escapes again by hopping a train, but during the night the Sarge catches up to him only to find that Joe's suitcase contains nothing but a bottle of champagne and wads of newspaper. They conclude that Dawn double-crossed Joe by repacking the suitcases while he was getting the car, and the Sarge proposes a plan to Joe to go after Dawn together. But, upon drinking a swallow of the champagne, the Sarge instantly goes into violent convulsions and falls down dead. The bottle was one of three that the Candy Man had filled with a solution of concentrated LSD to sneak them through customs earlier in the film. Joe then disembarks from the train and walks away, apparently betrayed by Dawn. An epilog shows Dawn in a sunny climate in the USA, joyfully driving a gleaming new yellow Corvette, and then later cuddling in bed with an unseen someone. The other suitcase is sitting near the bed, and Joe's bomber jacket hangs on the coat rack. Dawn smugly explains to the person she was certain the criminals wouldn't kill him and leave themselves with no way to get the money.

Years ago, a mob boss named Lucio Malatesta (George Touliatos) pinned the murder of rival Sammy Carboni (Gino Marrocco) on another rival named Angelo Allieghieri (Anthony Quinn), which led to Sammy's son Gianni vowing revenge.

Frankie Delano (Sylvester Stallone) has spent his life safeguarding Angelo as well as Angelo's daughter, Jennifer Barrett (Madeleine Stowe), whose unsavory husband Kip Barrett (Harry Van Gorkum) has had their young son Rawley (Ezra Perlman) placed in a boarding school against Jennifer's wishes. Jennifer was raised by her adoptive parents Whitney Towers (John Gilbert) and Peggy Towers (Dawn Greenhalgh) and is not aware that Angelo is her father.

After Angelo is killed in a restaurant by a hit man named Bruno (Billy Gardell), Frankie introduces himself, tells Jennifer who he is and what he has been doing.

A neurotic mess, Jennifer can barely handle the news that Kip is a philanderer, let alone the revelation that she is a gangster's daughter. But a DVD prepared by Angelo in the case of just such an event convinces Jennifer that it's the truth.

Jennifer certainly doesn't want a full-time bodyguard, even Frankie. She ditches Kip and then falls for Italian romance novelist Marcello (Raoul Bova), who lectures at her book club. Frankie has suspicions about Marcello, but his job is to stay on the sidelines.

Frankie rescues Jennifer from a string of attacks. With many of Angelo's enemies, including Lucio Malatesta, terminated, Frankie allows her to visit Italy with Marcello. But it turns out that Marcello is actually Gianni Carboni, who had Angelo killed. And now Gianni plans to kill Jennifer.

It's up to Frankie to protect her one more time.

Nick Conklin (Michael Douglas) is a skilled motorcyclist and a tough veteran New York City police officer facing possible criminal charges; Internal Affairs believes Nick was involved with his partner who was caught taking criminal money in a corruption scandal. Nick is divorced from his wife, who has custody of their two children. Nick also has financial difficulties due to alimony and child support as well as other concerns. Nick reports to a criminal investigation hearing being run by two officers from Internal Affairs, a conference that doesn't go well for him. They ask Nick about his involvement with several officers under investigation. When Nick refuses to squeal on his comrades, Internal Affairs threatens him, suggesting he's as corrupt as the others in the department. While having a drink at a local Italian restaurant/bar, Nick and his partner Charlie Vincent (Andy Garcia) observe two Japanese men having what appears to be a friendly lunch with some Italian gangsters. Nick is increasingly suspicious of the group until another Japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the Japanese. As the man turns to leave, one of the Japanese men at the table says, in Japanese, "The Oyabun [Godfather] will not stand for this." The leader of the Japanese group chimes in, "As always, such a troublesome child." The Japanese man finds these remarks insulting and he slashes the man's throat, stabs another in the chest, and then walks out. Nick and Charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills Nick in a nearby slaughterhouse. The suspect turns out to be a Yakuza gangster by the name of Sato (Yusaku Matsuda). The situation is further complicated when Nick's superior officer, Captain Oliver (John Spencer), tells him that Sato is to be extradited to Osaka and given to the police there. Nick is angry that Sato will not be tried for murder in the United States, but agrees to escort him to Japan. Nick's captain also has an ulterior motive for sending Nick overseas; he believes the excursion will keep Nick from causing more trouble and exacerbating the already biased Internal Affairs investigation of him. On the plane, Nick and Charlie talk about Nick's situation and how Nick's own expenses are beyond his means to pay them. At one point, while Charlie is out of his seat, Sato notices Nick cheating at solitaire and contemptibly chuckles to himself. Nick cruelly hits his prisoner in the mouth and lies about it when Charlie returns and asks what happened. When they arrive in Osaka, men identifying themselves as Japanese police immediately meet them on the plane, display a "transfer document" printed in Japanese and take Sato into their custody, leaving the plane by the rear exit. As Nick and Charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in English, indicating that the first "cops" were impostors. Nick and Charlie are taken to the headquarters of the Osaka Prefecture of Police a

nd questioned. They are also blamed for Sato's escape. After much haranguing by Nick (who shows xeno phobia) towards the Japanese, who rarely acknowledge that they can speak English, he and Charlie are allowed to observe the hunt for Sato. However, the senior police officer emphasizes that they have no authority in Japan and it is illegal for them to carry their guns, which are confiscated. They are assigned to Masahiro Matsumoto (Takakura), a mild-mannered and experienced officer, who will be their guide. Throughout the investigation Nick behaves rudely, offending Matsumoto, while Charlie tries to be more polite. Taken to a murder scene at a local nightclub, Nick recognizes the murder victim as one of the men at the airport who took Sato into custody. While the dead man is examined by forensic experts, one of them removes a \$100 bill from his mouth. Nick makes contact with an American blond nightclub hostess, Joyce (Kate Capshaw), who explains that the Japanese public, including the giggling hostesses in the club, all believe that Nick and Charlie are not to be taken seriously because they allowed Sato to easily escape from custody, and represent American inefficiency and stupidity. Through Joyce, Nick discovers that Sato is fighting a gang war with a notorious crime boss, Sugai (Tom Isaburo Wakayama). Sato used to be a lieutenant for Sugai and now wants his own territory to rule. Sato had traveled to New York to disrupt a meeting with American Italian gangsters about a scheme being set up by Sugai involving the package Sato had taken in the restaurant. Having joined a police raid of a gang hideout without permission, Nick takes some \$100 bills from a table, which he later shows are forgeries by burning one. The next day Matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to America; Nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in Matsumoto's superior's office. He suggests that the package stolen by Sato was either more samples of the forged bills or plates to make more. Late one night, after spending a few hours in a nightclub with Matsumoto, Nick and Charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from Matsumoto. They are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals Charlie's raincoat and lures Charlie into an underground parking garage. Nick follows, shouting for Charlie to come back, but is separated from his partner by a security gate. The unarmed Nick then watches in horror as Sato and several of his bszoku gang members briefly torture Charlie using swords and knives, before Sato beheads him. Distraught, Nick is comforted by Joyce at her apartment. Matsumoto arrives with Charlie's belongings, including his NYPD badge, which Nick gives to Matsumoto, and Charlie's service pistol, which Nick keeps for himself. Matsumoto and Nick trail one of Sato's operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and Nick admits to Matsumoto that he had taken some money in New York, where he says there is no "black-and-white" procedure, only "gray" areas. Matsumoto disagrees, saying "theft is theft" and that Nick's illegal action disgraces all police, including Matsumoto and Charlie. Nick realizes Matsumoto is right and humbly accepts Matsumoto's advice. In the morning the woman retrieves from a bank strongbox a sample counterfeit note (printed only on one side) which she passes to one of Sato's gang on the street. Nick and Matsumoto tail the man to a steel foundry where they find Sato meeting with Sugai, and discover that the package that Sato had stolen in New York contains one of the printing plates for the American \$100 bill. Nick intervenes when Sato leaves the meeting and a gunfight ensues. Sato escapes again when Nick is arrested by the swarming police for using a gun in public, and told he will be sent back to New York in disgrace. Nick boards the plane for New York but is able to sneak off to pursue Sato on his own. He finds that Matsumoto has been suspended and demoted by his police force, a deep humiliation. Joyce helps him meet Sugai, who explains that making counterfeit U.S. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of Hiroshima and the loss of dignity he and his family faced in the aftermath of World War II. Nick suggests a deal where Sugai can use Nick as an insignificant American to retrieve the stolen plate from Sato, leaving Sugai's reputation and hands clean. Sugai drops Nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. Nick is supplied with a shotgun. Sato arrives a short time later, as does Matsumoto. Matsumoto and Nick discover that Sato's men are planning a massacre. At the meeting table, Sato surrenders his single plate and requests recognition and his own territory. However, Sugai demands that Sato first atone for his offenses against the Yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. As he takes his position next to Sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between Sugai's and Sato's men. Sato escapes the fight on a dirt bike with Nick close behind. Nick is able to spill Sato off his bike and the two fight briefly, until Nick gains the advantage. The scene ends with Nick having to decide whether or not to kill Sato for Charlie and for all the humiliation he has suffered. The film ends with Matsumoto and Nick walking a handcuffed Sato into police HQ to the amazement of everyone and later receiving commendations, which Nick graciously accepts. At the airport, Nick thanks Matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. Underneath the shirt, Matsumoto finds the two counterfeit printing plates.

The film introduces a circle of youths who are addicted to playing Hellworld, an online computer game based on the Hellraiser series. The film opens at the funeral of Adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. The remaining five friends blame themselves for not having prevented Adam's suicide.

Two years later, they attend a private Hellworld Party at an old mansion after receiving invites through the game. Mike, Derrick and Allison are enthusiastic about the party, while Chelsea reluctantly accompanies them. Jake, who is still very much distressed by Adam's death, only agrees to show up after a female Hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. The quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by Philip Lemarchand), and provides them with cell phones to communicate with other guests.

As the party progresses, Allison, Derrick and Mike find themselves trapped in separate parts of the house, and are gruesomely killed by the Host, Pinhead, or Cenobite minions Chatterer II and Bound. Jake and Chelsea become mysteriously invisible to other party guests, and are stalked by the Host and the Cenobites.

Holing herself up in the attic, Chelsea finds items belonging to Adam, and discovers that the host is his father, who blames his son's friends for not helping break his addiction. Chelsea and Jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. The Host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilt.

y consciences. Before leaving, he lets Chelsea know that Allison, Derrick, and Mike have all perished in their respective caskets, and that only she and Jake remain alive. Chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from Chelsea's telephone. Looking towards the house, Chelsea sees Adam standing in the window.

Later, the Host sits in a bedroom, going through a suitcase containing Adam's possessions. He finds and opens the actual Lament Configuration, which summons the real Cenobites. Pinhead praises Adam's ingenuity and mocks the Host's disbelief before Chatterer and Bound tear him to pieces.

Jake and Chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the Host, who suddenly appears in the back seat. The two almost crash the car but are able to stop it. The last scene shows the police entering the bedroom in which the Host opened the box, the walls blood-smeared and the box lying on the floor.

Print the movie plots after removing HTML tags (if any)

In [53]:

```
#Function to clean html tags from a sentence
def removeHtml(sentence):
    pattern = re.compile('<.*?>')
    cleaned_text = re.sub(pattern, ' ',sentence)
    return cleaned_text

print(removeHtml(sent_1) + "\n")
print(removeHtml(sent_2) + "\n")
print(removeHtml(sent_3) + "\n")
print(removeHtml(sent_4) + "\n")
```

Set in Hamburg, West Germany, several criminals take advantage of the German privacy bank laws to use safe deposit boxes in a German bank to store large amounts of illicit cash. These include a Las Vegas mobster known only as the Attorney (Robert Webber) as well as a ruthless drug smuggler known as the Candy Man (Arthur Brauss) and a crooked overbearing U.S. Army sergeant (Scott Brady) and his meek-mannered partner the Major (Robert Stiles), who conspire on a big heroin and LSD smuggling score. Joe Collins (Warren Beatty), an American bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than \$1.5 million) with the help of Dawn Divine (Goldie Hawn), a hooker with a heart of gold. On the day of the robbery, Joe has Dawn phone in a bomb threat to the bank president, Mr. Kessel (Gert Fröbe), to create a diversion. Joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. The bank is closed and evacuated while Joe uses duplicate keys to empty the criminals' three safe deposit boxes into Dawn's large-size deposit box. (It is implied that Joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with Dawn.) Despite the fact that Kessel insists on burning through the wall to rescue Joe instead of waiting for the time lock to open, Joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. The next day, the three criminals, one by one, discover that their boxes are empty and they cannot complete their schemes or go to the police to report the thief. The Attorney flees the country while the others (Sarge, his partner the Major, and the Candy Man) search Dawn Divine's apartment as she was their common link and find clues that connect her to Joe. Sarge calls Kessel to get Joe's home address, but Joe is quickly tipped off by Kessel and he hurriedly sends Dawn to the train station with a suitcase packed with her take (\$765,000) promising to meet her later someplace out of the country. A long climatic chase begins as Dawn gives the Major the slip at the train station while the Candy Man and the Sarge chase Joe across a rail yard and through the Elbe Tunnel. Joe escapes on a car carrier truck, lugging his suitcase, but the Candy Man and the Sarge follow and catch up in the morning at a frozen lake in the countryside, where the Candy Man crashes a car through the ice and drowns while attempting to run Joe down with a stolen car. Joe escapes again by hopping a train, but during the night the Sarge catches up to him only to find that Joe's suitcase contains nothing but a bottle of champagne and wads of newspaper. They conclude that Dawn double-crossed Joe by repacking the suitcases while he was getting the car, and the Sarge proposes a plan to Joe to go after Dawn together. But, upon drinking a swallow of the champagne, the Sarge instantly goes into violent convulsions and falls down dead. The bottle was one of three that the Candy Man had filled with a solution of concentrated LSD to sneak them through customs earlier in the film. Joe then disembarks from the train and walks away, apparently betrayed by Dawn. An epilog shows Dawn in a sunny climate in the USA, joyfully driving a gleaming new yellow Corvette, and then later cuddling in bed with an unseen someone. The other suitcase is sitting near the bed, and Joe's bomber jacket hangs on the coat rack. Dawn smugly explains to the person she was certain the criminals wouldn't kill him and leave themselves with no way to get the money.

Years ago, a mob boss named Lucio Malatesta (George Touliatos) pinned the murder of rival Sammy Carbone (Gino Marrocco) on another rival named Angelo Allieghieri (Anthony Quinn), which led to Sammy's son Gianni vowing revenge.

Frankie Delano (Sylvester Stallone) has spent his life safeguarding Angelo as well as Angelo's daughter, Jennifer Barrett (Madeleine Stowe), whose unsavory husband Kip Barrett (Harry Van Gorkum) has had their young son Rawley (Ezra Perlman) placed in a boarding school against Jennifer's wishes. Jennifer was raised by her adoptive parents Whitney Towers (John Gilbert) and Peggy Towers (Dawn Greenthalgh) and is not aware that Angelo is her father.

After Angelo is killed in a restaurant by a hit man named Bruno (Billy Gardell), Frankie introduces himself, tells Jennifer who he is and what he has been doing.

A neurotic mess, Jennifer can barely handle the news that Kip is a philanderer, let alone the revelation that she is a gangster's daughter. But a DVD prepared by Angelo in the case of just such an event convinces Jennifer that it's the truth.

Jennifer certainly doesn't want a full-time bodyguard, even Frankie. She ditches Kip and then falls

for Italian romance novelist Marcello (Raoul Bova), who lectures at her book club. Frankie has suspicions about Marcello, but his job is to stay on the sidelines. Frankie rescues Jennifer from a string of attacks. With many of Angelo's enemies, including Lucio Ma latesta, terminated, Frankie allows her to visit Italy with Marcello. But it turns out that Marcello is actually Gianni Carboni, who had Angelo killed. And now Gianni plans to kill Jennifer. It's up to Frankie to protect her one more time.

Nick Conklin (Michael Douglas) is a skilled motorcyclist and a tough veteran New York City police officer facing possible criminal charges; Internal Affairs believes Nick was involved with his partner who was caught taking criminal money in a corruption scandal. Nick is divorced from his wife, who has custody of their two children. Nick also has financial difficulties due to alimony and child support as well as other concerns. Nick reports to a criminal investigation hearing being run by two officers from Internal Affairs, a conference that doesn't go well for him. They ask Nick about his involvement with several officers under investigation. When Nick refuses to squeal on his comrades, Internal Affairs threatens him, suggesting he's as corrupt as the others in the department. While having a drink at a local Italian restaurant/bar, Nick and his partner Charlie Vincent (Andy Garcia) observe two Japanese men having what appears to be a friendly lunch with some Italian gangsters. Nick is increasingly suspicious of the group until another Japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the Japanese. As the man turns to leave, one of the Japanese men at the table says, in Japanese, "The Oyabun [Godfather] will not stand for this." The leader of the Japanese group chimes in, "As always, such a troublesome child." The Japanese man finds these remarks insulting and he slashes the man's throat, stabs another in the chest, and then walks out. Nick and Charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills Nick in a nearby slaughterhouse. The suspect turns out to be a Yakuza gangster by the name of Sato (Yusaku Matsuda). The situation is further complicated when Nick's superior officer, Captain Oliver (John Spencer), tells him that Sato is to be extradited to Osaka and given to the police there. Nick is angry that Sato will not be tried for murder in the United States, but agrees to escort him to Japan. Nick's captain also has an ulterior motive for sending Nick overseas; he believes the excursion will keep Nick from causing more trouble and exacerbating the already biased Internal Affairs investigation of him. On the plane, Nick and Charlie talk about Nick's situation and how Nick's own expenses are beyond his means to pay them. At one point, while Charlie is out of his seat, Sato notices Nick cheating at solitaire and contemptibly chuckles to himself. Nick cruelly hits his prisoner in the mouth and lies about it when Charlie returns and asks what happened. When they arrive in Osaka, men identifying themselves as Japanese police immediately meet them on the plane, display a "transfer document" printed in Japanese and take Sato into their custody, leaving the plane by the rear exit. As Nick and Charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in English, indicating that the first "cops" were impostors. Nick and Charlie are taken to the headquarters of the Osaka Prefecture of Police and questioned. They are also blamed for Sato's escape. After much haranguing by Nick (who shows xenophobia) towards the Japanese, who rarely acknowledge that they can speak English, he and Charlie are allowed to observe the hunt for Sato. However, the senior police officer emphasizes that they have no authority in Japan and it is illegal for them to carry their guns, which are confiscated. They are assigned to Masahiro Matsumoto (Takakura), a mild-mannered and experienced officer, who will be their guide. Throughout the investigation Nick behaves rudely, offending Matsumoto, while Charlie tries to be more polite. Taken to a murder scene at a local nightclub, Nick recognizes the murder victim as one of the men at the airport who took Sato into custody. While the dead man is examined by forensic experts, one of them removes a \$100 bill from his mouth. Nick makes contact with an American blond nightclub hostess, Joyce (Kate Capshaw), who explains that the Japanese public, including the giggling hostesses in the club, all believe that Nick and Charlie are not to be taken seriously because they allowed Sato to easily escape from custody, and represent American inefficiency and stupidity. Through Joyce, Nick discovers that Sato is fighting a gang war with a notorious crime boss, Sugai (Tom Isaburo Wakayama). Sato used to be a lieutenant for Sugai and now wants his own territory to rule. Sato had traveled to New York to disrupt a meeting with American Italian gangsters about a scheme being set up by Sugai involving the package Sato had taken in the restaurant. Having joined a police raid of a gang hideout without permission, Nick takes some \$100 bills from a table, which he later shows are forgeries by burning one. The next day Matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to America; Nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in Matsumoto's superior's office. He suggests that the package stolen by Sato was either more samples of the forged bills or plates to make more. Late one night, after spending a few hours in a nightclub with Matsumoto, Nick and Charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from Matsumoto. They are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals Charlie's raincoat and lures Charlie into an underground parking garage. Nick follows, shouting for Charlie to come back, but is separated from his partner by a security gate. The unarmed Nick then watches in horror as Sato and several of his bszoku gang members briefly torture Charlie using swords and knives, before Sato beheads him. Distraught, Nick is comforted by Joyce at her apartment. Matsumoto arrives with Charlie's belongings, including his NYPD badge, which Nick gives to Matsumoto, and Charlie's service pistol, which Nick keeps for himself. Matsumoto and Nick trail one of Sato's operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and Nick admits to Matsumoto that he had taken some money in New York, where he says there is no "black-and-white" procedure, only "gray" areas. Matsumoto disagrees, saying "theft is theft" and that Nick's illegal action disgraces all police, including Matsumoto and Charlie. Nick realizes Matsumoto is right and humbly accepts Matsumoto's advice. In the morning the woman retrieves from a bank strongbox a sample counterfeit note (printed only on one side) which she passes to one of Sato's gang on the street. Nick and Matsumoto tail the man to a steel foundry where they find Sato meeting with Sugai, and discover that the package that Sato had stolen in New York contains one of the printing plates for the American \$100 bill. Nick intervenes when Sato leaves the meeting and a gunfight ensues. Sato escapes again when Nick is arrested by the swarming police for using a gun in public, and told he will be sent back to New York in disgrace. Nick boards the plane for New York but is able to sneak off to pursue Sato on his own. He finds that Matsumoto has been suspended and demoted by his police force, a deep humiliation. Joyce helps him meet Sugai, who explains that making counterfeit U.S. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of Hiroshima and the loss of dignity he and his family faced in the aftermath of W

orld War II. Nick suggests a deal where Sugai can use Nick as an insignificant American to retrieve the stolen plate from Sato, leaving Sugai's reputation and hands clean. Sugai drops Nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. Nick is supplied with a shotgun. Sato arrives a short time later, as does Matsumoto. Matsumoto and Nick discover that Sato's men are planning a massacre. At the meeting table, Sato surrenders his single plate and requests recognition and his own territory. However, Sugai demands that Sato first atone for his offenses against the Yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. As he takes his position next to Sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between Sugai's and Sato's men. Sato escapes the fight on a dirt bike with Nick close behind. Nick is able to spill Sato off his bike and the two fight briefly, until Nick gains the advantage. The scene ends with Nick having to decide whether or not to kill Sato for Charlie and for all the humiliation he has suffered. The film ends with Matsumoto and Nick walking a handcuffed Sato into police HQ to the amazement of everyone and later receiving commendations, which Nick graciously accepts. At the airport, Nick thanks Matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. Underneath the shirt, Matsumoto finds the two counterfeit printing plates.

The film introduces a circle of youths who are addicted to playing Hellworld, an online computer game based on the Hellraiser series. The film opens at the funeral of Adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. The remaining five friends blame themselves for not having prevented Adam's suicide.

Two years later, they attend a private Hellworld Party at an old mansion after receiving invites through the game. Mike, Derrick and Allison are enthusiastic about the party, while Chelsea reluctantly accompanies them. Jake, who is still very much distressed by Adam's death, only agrees to show up after a female Hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. The quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by Philip Lemarchand), and provides them with cell phones to communicate with other guests.

As the party progresses, Allison, Derrick and Mike find themselves trapped in separate parts of the house, and are gruesomely killed by the Host, Pinhead, or Cenobite minions Chatterer II and Bound. Jake and Chelsea become mysteriously invisible to other party guests, and are stalked by the Host and the Cenobites.

Holing herself up in the attic, Chelsea finds items belonging to Adam, and discovers that the host is his father, who blames his son's friends for not helping break his addiction. Chelsea and Jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. The Host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilty consciences. Before leaving, he lets Chelsea know that Allison, Derrick, and Mike have all perished in their respective caskets, and that only she and Jake remain alive. Chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from Chelsea's telephone. Looking towards the house, Chelsea sees Adam standing in the window.

Later, the Host sits in a bedroom, going through a suitcase containing Adam's possessions. He finds and opens the actual Lament Configuration, which summons the real Cenobites. Pinhead praises Adam's ingenuity and mocks the Host's disbelief before Chatterer and Bound tear him to pieces.

Jake and Chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the Host, who suddenly appears in the back seat. The two almost crash the car but are able to stop it. The last scene shows the police entering the bedroom in which the Host opened the box, the walls blood-smeared and the box lying on the floor.

Print the movie plots after de-contracting the movies

In [54]:

```
# https://stackoverflow.com/a/47091490/4084039
# https://en.wikipedia.org/wiki/Wikipedia:List_of_English_contractions
import re

#Expand the movie plots x is an input string of any length. Convert all the words to lower case
def decontracted(x):
    x = str(x).lower()
    x = x.replace(",000,000", " m").replace(",000", " k").replace("'", "'").replace('"', '"')\
        .replace("won't", " will not").replace("cannot", " can not").replace("can't", " can no
t")\
        .replace("n't", " not").replace("what's", " what is").replace("it's", " it is")\
        .replace("'ve", " have").replace("i'm", " am").replace("re", " are")\
        .replace("he's", " he is").replace("she's", " she is").replace("'s", " own")\
        .replace("%", " percent ").replace("₹", " rupee ").replace("$", " dollar ") \
        .replace("€", " euro ").replace("ll", " will").replace("how's", " how has").replace("y
'all", " you all")\
        .replace("o'clock", " of the clock").replace("ne'er", " never").replace("let's", " let us
")\
        .replace("finna", " fixing to").replace("gonna", " going to").replace("gimme", " give me")
).replace("gotta", " got to").replace("d", " would")\
        .replace("daresn't", " dare not").replace("dasn't", " dare not").replace("e'er", " ever")
.replace("everyone's", " everyone is")\
        .replace("'cause", " because")

    x = re.sub(r"([0-9]+)000000", r"\1m", x)
    x = re.sub(r"([0-9]+)000", r"\1k", x)
    return x

print(decontracted(sent_1) + "\n")
print(decontracted(sent_2) + "\n")
print(decontracted(sent_3) + "\n")
print(decontracted(sent_4) + "\n")
```

set in hamburg, west germany, several criminals take advantage of the german privacy bank laws to use safe deposit boxes in a german bank to store large amounts of illicit cash. these include a las vegas mobster known only as the attorney (robert webber) as well as a ruthless drug smuggler known as the candy man (arthur brauss) and a crooked overbearing u.s. army sergeant (scott brady) and his meek-mannered partner the major (robert stiles), who conspire on a big heroin and lsd smuggling score. joe collins (warren beatty), an american bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than dollar 1.5 million) with the help of dawn divine (goldie hawn), a hooker with a heart of gold. on the day of the robbery, joe has dawn phone in a bomb threat to the bank president, mr. kessel (gert fröbe), to create a diversion. joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. the bank is closed and evacuated while joe uses duplicate keys to empty the criminals' three safe deposit boxes into dawn's own large-size deposit box. (it is implied that joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with dawn.) despite the fact that kessel insists on burning through the wall to rescue joe instead of waiting for the time lock to open, joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. the next day, the three criminals, one by one, discover that their boxes are empty and they can not complete their schemes or go to the police to report the thief. the attorney flees the country while the others (sarge, his partner the major, and the candy man) search dawn divine's own apartment as she was their common link and find clues that connect her to joe. sarge calls kessel to get joe's own home address, but joe is quickly tipped off by kessel and he hurriedly sends dawn to the train station with a suitcase packed with her take (dollar 765 k) promising to meet her later someplace out of the country. a long climatic chase begins as dawn gives the major the slip at the train station while the candy man and the sarge chase joe across a rail yard and through the elbe tunnel. joe escapes on a car carrier truck, lugging his suitcase, but the candy man and the sarge follow and catch up in the morning at a frozen lake in the countryside, where the candy man crashes a car through the ice and drowns while attempting to run joe down with a stolen car. joe escapes again by hopping a train, but during the night the sarge catches up to him only to find that joe's own suitcase contains nothing but a bottle of champagne and wads of newspaper. they conclude that dawn double-crossed joe by repacking the suitcases while he was getting the car, and the sarge proposes a plan to joe to go after dawn together. but, upon drinking a swallow of the champagne, the sarge instantly goes into violent convulsions and falls down dead. the bottle was one of three that the candy man had filled with a solution of concentrated lsd to sneak them through customs earlier in the film. joe then disembarks from the train and walks away, apparently betrayed by dawn. an epilogue shows dawn in a sunny climate in the usa, joyfully driving a gleaming new yellow corvette, and then later cuddling in bed with an unseen someone. the other suitcase is sitting near the bed, and joe's own bomber jacket hangs on the coat rack. dawn smugly explains to the person she was certain the criminals would not kill him and leave themselves with no way to get the money.

years ago, a mob boss named lucio malatesta (george touliatos) pinned the murder of rival sammy carbone (gino marrocco) on another rival named angelo allieghieri (anthony quinn), which led to sammy's own son gianni vowing revenge. frankie delano (sylvestor stallone) has spent his life safeguarding angelo as well as angelo's own daughter, jennifer barrett (madeleine stowe), whose unsavory husband kip barrett (harry van gorkum) has had their young son rawley (ezra perlman) placed in a boarding school against jennifer's own wishes. jennifer was raised by her adoptive parents whitney towers (john gilbert) and peggy towers (dawn greenhalgh) and is not aware that angelo is her father. after angelo is killed in a restaurant by a hit man named bruno (billy gardell), frankie introduces himself, tells jennifer who he is and what he has been doing.

a neurotic mess, jennifer can barely handle the news that kip is a philanderer, let alone the revelation that she is a gangster's own daughter. but a dvd prepared by angelo in the case of just such an event convinces jennifer that it is the truth. jennifer certainly does not want a full-time bodyguard, even frankie. she ditches kip and then falls for italian romance novelist marcello (raoul bova), who lectures at her book club. frankie has suspicions about marcello, but his job is to stay on the sidelines. frankie rescues jennifer from a string of attacks. with many of angelo's own enemies, including lucio malatesta, terminated, frankie allows her to visit italy with marcello. but it turns out that marcello is actually gianni carboni, who had angelo killed. and now gianni plans to kill jennifer. it is up to frankie to protect her one more time.

nick conklin (michael douglas) is a skilled motorcyclist and a tough veteran new york city police officer facing possible criminal charges; internal affairs believes nick was involved with his partner who was caught taking criminal money in a corruption scandal. nick is divorced from his wife, who has custody of their two children. nick also has financial difficulties due to alimony and child support as well as other concerns. nick reports to a criminal investigation hearing being run by two officers from internal affairs, a conference that does not go well for him. they ask nick about his involvement with several officers under investigation. when nick refuses to squeal on his comrades, internal affairs threatens him, suggesting he is as corrupt as the others in the department. while having a drink at a local italian restaurant/bar, nick and his partner charlie vincent (andy garcia) observe two japanese men having what appears to be a friendly lunch with some italian gangsters. nick is increasingly suspicious of the group until another japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the japanese. as the man turns to leave, one of the japanese men at the table says, in japanese, "the oyabun [godfather] will not stand for this." the leader of the japanese group chimes in, "as always, such a troublesome child." the japanese man finds these remarks insulting and he slashes the man's own throat, stabs another in the chest, and then walks out. nick and charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills nick in a nearby slaughterhouse. the suspect turns out to be a yakuza gangster by the name of sato (yusaku matsuda). the situation is further complicated when nick's own superior officer, captain oliver (john spencer), tells him that sato is to be extradited to osaka and given to the police there. nick is angry that sato will not be tried for murder in the united states, but agrees to escort him to japan. nick's captain also has an ulterior motive for sending nick overseas; he believes the excursion will keep nick from causing more trouble and exacerbating the already biased internal affairs investigation of him. on the plane, nick and charlie talk about nick's own situation and how nick's own expenses are beyond his means to pay them. at one point, while charlie is out of his seat, sato notices nick cheating at solitaire and contemptibly chuckles to himself. nick cruelly hits his prisoner in the mouth and lies about it when charlie returns and asks what happened. when they arrive in osaka, men identifying themselves as japanese police immediately meet them on the plane, display a "transfer document" printed in japanese and take sato into their custody, leaving the plane by the rear exit. as nick and charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in english, indicating that the first "cops" were impostors. nick and charlie are taken to the headquarters of the osaka prefecture of police and questioned. they are also blamed for sato's own escape. after much haranguing by nick (who shows xenophobia) towards the japanese, who rarely acknowledge that they can speak english, he and charlie are allowed to observe the hunt for sato. however, the senior police officer emphasizes that they have no authority in japan and it is illegal for them to carry their guns, which are confiscated. they are assigned to masahiro matsumoto (takakura), a mild-mannered and experienced officer, who will be their guide. throughout the investigation nick behaves rudely, offending matsumoto, while charlie tries to be more polite. taken to a murder scene at a local nightclub, nick recognizes the murder victim as one of the men at the airport who took sato into custody. while the dead man is examined by forensics experts, one of them removes a dollar 100 bill from his mouth. nick makes contact with an american blond nightclub hostess, joyce (kate capshaw), who explains that the japanese public, including the giggling hostesses in the club, all believe that nick and charlie are not to be taken seriously because they allowed sato to easily escape from custody, and represent american inefficiency and stupidity. through joyce, nick discovers that sato is fighting a gang war with a notorious crime boss, sugai (tomisaburo wakayama). sato used to be a lieutenant for sugai and now wants his own territory to rule. sato had traveled to new york to disrupt a meeting with an american italian gangster about a scheme being set up by sugai involving the package sato had taken in the restaurant. having joining a police raid of a gang hideout without permission, nick takes some dollar 100 bills from a table, which he later shows are forgeries by burning one. the next day matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to america; nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in matsumoto's own superior's own office. he suggests that the package stolen by sato was either more samples of the forged bills or plates to make more. late one night, after spending a few hours in a nightclub with matsumoto, nick and charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from matsumoto. they are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals charlie's raincoat and lures charlie into an underground parking garage. nick follows, shouting for charlie to come back, but is separated from his partner by a security gate. the unarmed nick then watches in horror as sato and several of his bszoku gang members briefly torture charlie using swords and knives, before sato beheads him. distraught, nick is comforted by joyce at her apartment. matsumoto arrives with charlie's own belongings, including his nypd badge, which nick gives to matsumoto, and charlie's own service pistol, which nick keeps for himself. matsumoto and nick trail one of sato's own operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and nick admits to matsumoto that he had taken some money in new york, where he says there is no "black-and-white" procedure, only "gray" areas. matsumoto disagrees, saying "theft is theft" and that nick's own illegal action disgraces all police, including matsumoto and charlie. nick realizes matsumoto is right and humbly accepts matsumoto's own advice. in the morning the woman retrieves from a bank's strongbox a sample counterfeit note (printed only on one side) which she passes to one of sato's own gang on the street. nick and matsumoto tail the man to a steel foundry where they find sato meeting with sugai, and discover that the package that sato had stolen in new york contains one of the printing plates for the american dollar 100 bill. nick intervenes when sato leaves the meeting and a gunfight ensues. sato escapes again when nick is arrested by the swarming police for using a gun in public, and told he will be sent back to new york.

ork in disgrace. nick boards the plane for new york but is able to sneak off to pursue sato on his own. he finds that matsumoto has been suspended and demoted by his police force, a deep humiliation. j oyce helps him meet sugai, who explains that making counterfeit u.s. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of hiroshima and the loss of dignity he and his family faced in the aftermath of world war ii. nick suggests a deal where sugai can use nick as an insignificant american to retrieve the stolen plate from sato, leaving sugai own reputation and hands clean. sugai drops nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. nick is supplied with a shotgun. sato arrives a short time later, as does matsumoto. matsumoto and nick discover that sato own men are planning a massacre. at the meeting table, sato surrenders his single plate and requests recognition and his own territory. however, sugai demands that sato first atone for his offenses against the yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. as he takes his position next to sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between sugai own and sato own men. sato escapes the fight on a dirt bike with nick close behind. nick is able to spill sato off his bike and the two fight briefly, until nick gains the advantage. the scene ends with nick having to decide whether or not to kill sato for charlie and for all the humiliation he has suffered. the film ends with matsumoto and nick walking a handcuffed sato into police hq to the amazement of everyone and later receiving commendations, which nick graciously accepts. at the airport, nick thanks matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. underneath the shirt, matsumoto finds the two counterfeit printing plates.

the film introduces a circle of youths who are addicted to playing hellworld, an online computer game based on the hellraiser series. the film opens at the funeral of adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. the remaining five friends blame themselves for not having prevented adam own suicide.

two years later, they attend a private hellworld party at an old mansion after receiving invites through the game. mike, derrick and allison are enthusiastic about the party, while chelsea reluctantly accompanies them. jake, who is still very much distressed by adam own death, only agrees to show up after a female hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. the quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by philip lemarchand), and provides them with cell phones to communicate with other guests.

as the party progresses, allison, derrick and mike find themselves trapped in separate parts of the house, and are gruesomely killed by the host, pinhead, or cenobite minions chatterer ii and bound. jake and chelsea become mysteriously invisible to other party guests, and are stalked by the host and the cenobites.

holding herself up in the attic, chelsea finds items belonging to adam, and discovers that the host is his father, who blames his son own friends for not helping break his addiction. chelsea and jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. the host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilty consciences. before leaving, he lets chelsea know that allison, derrick, and mike have all perished in their respective caskets, and that only she and jake remain alive. chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from chelsea own telephone. looking towards the house, chelsea sees adam standing in the window.

later, the host sits in a bedroom, going through a suitcase containing adam own possessions. he finds and opens the actual lament configuration, which summons the real cenobites. pinhead praises adam own ingenuity and mocks the host own disbelief before chatterer and bound tear him to pieces.

jake and chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the host, who suddenly appears in the back seat. the two almost crash the car but are able to stop it. the last scene shows the police entering the bedroom in which the host opened the box, the walls blood-smeared and the box lying on the floor.

Remove words with numbers

In [55]:

```
#remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
```

```
'''
>>> import re
>>> s = "ABCD abcd AB55 55CD A55D 5555"
>>> re.sub("\S*\d\S*", "", s).strip()

'ABCD abcd'
>>>'''

sent_1 = re.sub("\S*\d\S*", " ", sent_1).strip()
print(sent_1 + "\n")

sent_2 = re.sub("\S*\d\S*", " ", sent_2).strip()
print(sent_2 + "\n")

sent_3 = re.sub("\S*\d\S*", " ", sent_3).strip()
print(sent_3 + "\n")

sent_4 = re.sub("\S*\d\S*", " ", sent_4).strip()
print(sent_4 + "\n")
```

Set in Hamburg. West Germanv. several criminals take advantage of the German privacy bank laws to us

Joe and Hamburg, both Germany, several criminals take advantage of the German privacy laws to use the safe deposit boxes in a German bank to store large amounts of illicit cash. These include a Las Vegas mobster known only as the Attorney (Robert Webber) as well as a ruthless drug smuggler known as the Candy Man (Arthur Brauss) and a crooked overbearing U.S. Army sergeant (Scott Brady) and his meek-mannered partner the Major (Robert Stiles), who conspire on a big heroin and LSD smuggling score. Joe Collins (Warren Beatty), an American bank security consultant, has been spying on them and makes mysterious and elaborate preparations to steal their money (totaling more than \$1 million) with the help of Dawn Divine (Goldie Hawn), a hooker with a heart of gold. On the day of the robbery, Joe has Dawn phone in a bomb threat to the bank president, Mr. Kessel (Gert Fröbe), to create a diversion. Joe locks himself inside the bank vault with a gold bar normally displayed in the lobby to supposedly save it. The bank is closed and evacuated while Joe uses duplicate keys to empty the criminals' three safe deposit boxes into Dawn's large-size deposit box. (It is implied that Joe had obtained the necessary bank info and secretly copied the criminals' keys while they were engaged in sexual trysts with Dawn.) Despite the fact that Kessel insists on burning through the wall to rescue Joe instead of waiting for the time lock to open, Joe succeeds in the heist and is hailed as a hero for "preventing" the robbery of the gold bar. The next day, the three criminals, one by one, discover that their boxes are empty and they cannot complete their schemes or go to the police to report the thief. The Attorney flees the country while the others (Sarge, his partner the Major, and the Candy Man) search Dawn Divine's apartment as she was their common link and find clues that connect her to Joe. Sarge calls Kessel to get Joe's home address, but Joe is quickly tipped off by Kessel and he hurriedly sends Dawn to the train station with a suitcase packed with her take, promising to meet her later someplace out of the country. A long climatic chase begins as Dawn gives the Major the slip at the train station while the Candy Man and the Sarge chase Joe across a rail yard and through the Elbe Tunnel. Joe escapes on a car carrier truck, lugging his suitcase, but the Candy Man and the Sarge follow and catch up in the morning at a frozen lake in the countryside, where the Candy Man crashes a car through the ice and drowns while attempting to run Joe down with a stolen car. Joe escapes again by hopping a train, but during the night the Sarge catches up to him only to find that Joe's suitcase contains nothing but a bottle of champagne and wads of newspaper. They conclude that Dawn double-crossed Joe by repacking the suitcases while he was getting the car, and the Sarge proposes a plan to Joe to go after Dawn together. But, upon drinking a swallow of the champagne, the Sarge instantly goes into violent convulsions and falls down dead. The bottle was one of three that the Candy Man had filled with a solution of concentrated LSD to sneak them through customs earlier in the film. Joe then disembarks from the train and walks away, apparently betrayed by Dawn. An epilog shows Dawn in a sunny climate in the USA, joyfully driving a gleaming new yellow Corvette, and then later cuddling in bed with an unseen someone. The other suitcase is sitting near the bed, and Joe's bomber jacket hangs on the coat rack. Dawn smugly explains to the person she was certain the criminals wouldn't kill him and leave themselves with no way to get the money.

Years ago, a mob boss named Lucio Malatesta (George Touliatos) pinned the murder of rival Sammy Carboni (Gino Marrocco) on another rival named Angelo Allieghieri (Anthony Quinn), which led to Sammy's son Gianni vowing revenge.

Frankie Delano (Sylvester Stallone) has spent his life safeguarding Angelo as well as Angelo's daughter, Jennifer Barrett (Madeleine Stowe), whose unsavory husband Kip Barrett (Harry Van Gorkum) has had their young son Rawley (Ezra Perlman) placed in a boarding school against Jennifer's wishes. Jennifer was raised by her adoptive parents Whitney Towers (John Gilbert) and Peggy Towers (Dawn Greenthalgh) and is not aware that Angelo is her father.

After Angelo is killed in a restaurant by a hit man named Bruno (Billy Gardell), Frankie introduces himself, tells Jennifer who he is and what he has been doing.

A neurotic mess, Jennifer can barely handle the news that Kip is a philanderer, let alone the revelation that she is a gangster's daughter. But a DVD prepared by Angelo in the case of just such an event convinces Jennifer that it's the truth.

Jennifer certainly doesn't want a full-time bodyguard, even Frankie. She ditches Kip and then falls for Italian romance novelist Marcello (Raoul Bova), who lectures at her book club. Frankie has suspicions about Marcello, but his job is to stay on the sidelines.

Frankie rescues Jennifer from a string of attacks. With many of Angelo's enemies, including Lucio Malatesta, terminated, Frankie allows her to visit Italy with Marcello. But it turns out that Marcello is actually Gianni Carboni, who had Angelo killed. And now Gianni plans to kill Jennifer.

It's up to Frankie to protect her one more time.

Nick Conklin (Michael Douglas) is a skilled motorcyclist and a tough veteran New York City police officer facing possible criminal charges; Internal Affairs believes Nick was involved with his partner who was caught taking criminal money in a corruption scandal. Nick is divorced from his wife, who has custody of their two children. Nick also has financial difficulties due to alimony and child support as well as other concerns. Nick reports to a criminal investigation hearing being run by two officers from Internal Affairs, a conference that doesn't go well for him. They ask Nick about his involvement with several officers under investigation. When Nick refuses to squeal on his comrades, Internal Affairs threatens him, suggesting he's as corrupt as the others in the department. While having a drink at a local Italian restaurant/bar, Nick and his partner Charlie Vincent (Andy Garcia) observe two Japanese men having what appears to be a friendly lunch with some Italian gangsters. Nick is increasingly suspicious of the group until another Japanese man enters the restaurant with several armed henchmen and seizes a small package at gunpoint from the leader of the Japanese. As the man turns to leave, one of the Japanese men at the table says, in Japanese, "The Oyabun [Godfather] will not stand for this." The leader of the Japanese group chimes in, "As always, such a troublesome child." The Japanese man finds these remarks insulting and he slashes the man's throat, stabs another in the chest, and then walks out. Nick and Charlie follow immediately and, after a short chase, arrest the suspect after he nearly kills Nick in a nearby slaughterhouse. The suspect turns out to be a Yakuza gangster by the name of Sato (Yusaku Matsuda). The situation is further complicated when Nick's superior officer, Captain Oliver (John Spencer), tells him that Sato is to be extradited to Osaka and given to the police there. Nick is angry that Sato will not be tried for murder in the United States, but agrees to escort him to Japan. Nick's captain also has an ulterior motive for sending Nick overseas; he believes the excursion will keep Nick from causing more trouble and exacerbating the already biased Internal Affairs investigation of him. On the plane, Nick and Charlie talk about Nick's situation and how Nick's own expenses are beyond his means to pay them. At one point, while Charlie is out of his seat, Sato notices Nick cheating at solitaire and contemptibly chuckles to himself. Nick crue

lly hits his prisoner in the mouth and lies about it when Charlie returns and asks what happened. When they arrive in Osaka, men identifying themselves as Japanese police immediately meet them on the plane, display a "transfer document" printed in Japanese and take Sato into their custody, leaving the plane by the rear exit. As Nick and Charlie are about to get off the plane themselves, another group of police enter from the front and identify themselves in English, indicating that the first "cops" were impostors. Nick and Charlie are taken to the headquarters of the Osaka Prefecture of Police and questioned. They are also blamed for Sato's escape. After much haranguing by Nick (who shows xenophobia) towards the Japanese, who rarely acknowledge that they can speak English, he and Charlie are allowed to observe the hunt for Sato. However, the senior police officer emphasizes that they have no authority in Japan and it is illegal for them to carry their guns, which are confiscated. They are assigned to Masahiro Matsumoto (Takakura), a mild-mannered and experienced officer, who will be their guide. Throughout the investigation Nick behaves rudely, offending Matsumoto, while Charlie tries to be more polite. Taken to a murder scene at a local nightclub, Nick recognizes the murder victim as one of the men at the airport who took Sato into custody. While the dead man is examined by forensic experts, one of them removes a bill from his mouth. Nick makes contact with an American blond nightclub hostess, Joyce (Kate Capshaw), who explains that the Japanese public, including the giggling hostesses in the club, all believe that Nick and Charlie are not to be taken seriously because they allowed Sato to easily escape from custody, and represent American inefficiency and stupidity. Through Joyce, Nick discovers that Sato is fighting a gang war with a notorious crime boss, Sugai (Tomisaburo Wakayama). Sato used to be a lieutenant for Sugai and now wants his own territory to rule. Sato had traveled to New York to disrupt a meeting with American Italian gangsters about a scheme being set up by Sugai involving the package Sato had taken in the restaurant. Having joined a police raid on a gang hideout without permission, Nick takes some bills from a table, which he later shows are forgeries by burning one. The next day Matsumoto explains they have dishonoured themselves, him and the police force by this theft, which has been reported back to America; Nick just claims he ought not to have "snitched" to his superiors, and demonstrates the forgery in Matsumoto's superior's office. He suggests that the package stolen by Sato was either more samples of the forged bills or plates to make more. Late one night, after spending a few hours in a nightclub with Matsumoto, Nick and Charlie walk back to their hotel slightly drunk and unescorted, despite previous warnings about their safety from Matsumoto. They are harassed by a young punk on a motorcycle, and it seems to be a joke until the motorcyclist steals Charlie's raincoat and lures Charlie into an underground parking garage. Nick follows, shouting for Charlie to come back, but is separated from his partner by a security gate. The unarmed Nick then watches in horror as Sato and several of his bszoku gang members briefly torture Charlie using swords and knives, before Sato beheads him. Distraught, Nick is comforted by Joyce at her apartment. Matsumoto arrives with Charlie's belongings, including his NYPD badge, which Nick gives to Matsumoto, and Charlie's service pistol, which Nick keeps for himself. Matsumoto and Nick trail one of Sato's operatives, a well-dressed young woman; overnight the policemen discuss their different cultures, and Nick admits to Matsumoto that he had taken some money in New York, where he says there is no "black-and-white" procedure, only "gray" areas. Matsumoto disagrees, saying "theft is theft" and that Nick's illegal action disgraces all police, including Matsumoto and Charlie. Nick realizes Matsumoto is right and humbly accepts Matsumoto's advice. In the morning the woman retrieves from a bank strongbox a sample counterfeit note (printed only on one side) which she passes to one of Sato's gang on the street. Nick and Matsumoto tail the man to a steel foundry where they find Sato meeting with Sugai, and discover that the package that Sato had stolen in New York contains one of the printing plates for the American bill. Nick intervenes when Sato leaves the meeting and a gunfight ensues. Sato escapes again when Nick is arrested by the swarming police for using a gun in public, and told he will be sent back to New York in disgrace. Nick boards the plane for New York but is able to sneak off to pursue Sato on his own. He finds that Matsumoto has been suspended and demoted by his police force, a deep humiliation. Joyce helps him meet Sugai, who explains that making counterfeit U.S. currency is his revenge for the pollution, the "black rain", that he witnessed after the bombing of Hiroshima and the loss of dignity he and his family faced in the aftermath of World War II. Nick suggests a deal where Sugai can use Nick as an insignificant American to retrieve the stolen plate from Sato, leaving Sugai's reputation and hands clean. Sugai drops Nick at the outskirts of a remote farm where a meeting of the oyabun, the other crime bosses of the region, is to take place. Nick is supplied with a shotgun. Sato arrives a short time later, as does Matsumoto. Matsumoto and Nick discover that Sato's men are planning a massacre. At the meeting table, Sato surrenders his single plate and requests recognition and his own territory. However, Sugai demands that Sato first atone for his offenses against the Yakuza code in the traditional way: he is ordered to cut off one of his fingers (yubitsume), which he duly does. As he takes his position next to Sugai, he stabs the elder gangster in the hand and escapes with both the plates, prompting a gunfight between Sugai's and Sato's men. Sato escapes the fight on a dirt bike with Nick close behind. Nick is able to spill Sato off his bike and the two fight briefly, until Nick gains the advantage. The scene ends with Nick having to decide whether or not to kill Sato for Charlie and for all the humiliation he has suffered. The film ends with Matsumoto and Nick walking a handcuffed Sato into police HQ to the amazement of everyone and later receiving commendations, which Nick graciously accepts. At the airport, Nick thanks Matsumoto for his assistance and his friendship, and gives him a gift box containing a dress shirt. Underneath the shirt, Matsumoto finds the two counterfeit printing plates.

The film introduces a circle of youths who are addicted to playing Hellworld, an online computer game based on the Hellraiser series. The film opens at the funeral of Adam, one of their friends who was obsessed with the game and ultimately committed suicide after becoming too immersed in the game. The remaining five friends blame themselves for not having prevented Adam's suicide.

Two years later, they attend a private Hellworld Party at an old mansion after receiving invites through the game. Mike, Derrick and Allison are enthusiastic about the party, while Chelsea reluctantly accompanies them. Jake, who is still very much distressed by Adam's death, only agrees to show up after a female Hellworld player with whom he has struck up an online friendship asks him to attend so they can meet. The quintet are cordially welcomed by the middle-aged party host, who offers them drinks, shows them around the mansion (allegedly a former convent and asylum also built by Philip Lemarchand), and provides them with cell phones to communicate with other guests.

As the party progresses, Allison, Derrick and Mike find themselves trapped in separate parts of the house, and are gruesomely killed by the Host, Pinhead, or Cenobite minions Chatterer II and Bound. Jake and Chelsea become mysteriously invisible to other party guests, and are stalked by the Host and the Cenobites.

Holing herself up in the attic, Chelsea finds items belonging to Adam, and discovers that the host is his father, who blames his son's friends for not helping break his addiction. Chelsea and Jake try to flee, only to discover that they have been buried alive and are receiving messages from the host via cell phones in their respective caskets. The Host informs them that they are just coming out of a hallucination induced by a powerful psychedelic he exposed them to upon their arrival, and that the events they have been experiencing have been the result of hypnotic suggestion and their own guilty consciences. Before leaving, he lets Chelsea know that Allison, Derrick, and Mike have all perished in their respective caskets, and that only she and Jake remain alive. Chelsea begins to slip into another hallucination when she is abruptly pulled above ground by police and paramedics, who say they were informed by a phone call from Chelsea's telephone. Looking towards the house, Chelsea sees Adam standing in the window.

Later, the Host sits in a bedroom, going through a suitcase containing Adam's possessions. He finds and opens the actual Lament Configuration, which summons the real Cenobites. Pinhead praises Adam's ingenuity and mocks the Host's disbelief before Chatterer and Bound tear him to pieces. Jake and Chelsea are shown driving into the sunrise, when they receive a mysterious phone call from the Host, who suddenly appears in the back seat. The two almost crash the car but are able to stop it. The last scene shows the police entering the bedroom in which the Host opened the box, the walls blood-smeared and the box lying on the floor.

Utility functions to clean the movie synopses

In [2]:

```
#Remove words with numbers python: https://stackoverflow.com/a/18082370/4084039
def removeNumbers(sentence):
    sentence = re.sub("\S*\d\S*", " ", sentence).strip()
    return (sentence)

#Function to clean html tags from a sentence
def removeHtml(sentence):
    pattern = re.compile('<.*?>')
    cleaned_text = re.sub(pattern, ' ',sentence)
    return cleaned_text

#Remove URL from sentences.
def removeURL(sentence):
    text = re.sub(r"http\S+", " ", sentence)
    sentence = re.sub(r"www.\S+", " ", text)
    return (sentence)

#Function to keep only words containing letters A-Z and a-z. This will remove all punctuations, special characters etc. https://stackoverflow.com/a/5843547/4084039
def removePunctuations(sentence):
    cleaned_text = re.sub('[^a-zA-Z]', ' ',sentence)
    return (cleaned_text)

#https://stackoverflow.com/questions/37012948/regex-to-match-an-entire-word-that-contains-repeated-character
#Remove words like 'zzzzzzzzzzzzzzzzzzzzzzzz', 'testtting', 'grrrrrrreeeetttt' etc. Preserves words like 'looks', 'goods', 'soon' etc. We will remove all such words which has three consecutive repeating characters.
def removePatterns(sentence):
    cleaned_text = re.sub("\s*\b(?:=\w*(\w)\1{2,})\w*\b", ' ',sentence)
    return (cleaned_text)

#Stemming and stopwords removal
from nltk.stem.snowball import SnowballStemmer
sno = SnowballStemmer(language='english')

#Removing the word 'not' from stopwords
default_stopwords = set(stopwords.words('english'))
remove_not = set(['no', 'nor', 'not'])
custom_stopwords = default_stopwords - remove_not

print(custom_stopwords)

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

In [57]:

```
# Combining all the above data cleaning methodologies as discussed above.
string=' '
stemmed_word=' '

preprocessed_movie_plots =[]

for movie_plot in tqdm(dataframe['plot_synopsis'].values):
    filtered_sentence=[]
    movie_plot = decontracted(movie_plot)
    movie_plot = removeNumbers(movie_plot)
    movie_plot = removeHtml(movie_plot)
    movie_plot = removeURL(movie_plot)
    movie_plot = removePunctuations(movie_plot)
    movie_plot = removePatterns(movie_plot)

    for cleaned_words in movie_plot.split():
        if((cleaned_words not in custom_stopwords) and (len(cleaned_words)>2)):
            stemmed_word=(sno.stem(cleaned_words.lower()))
            filtered_sentence.append(stemmed_word)
        else:
            continue
    movie_plot = " ".join(filtered_sentence) #Final string of cleaned words
    preprocessed_movie_plots.append(movie_plot.strip()) #Data corpus contaning cleaned movie_plots from the whole
dataset

#Adding a column of CleanedPlots to the table final which stores the data_corpus after pre-processing the movie_p
lots
dataframe['CleanedPlots']=preprocessed_movie_plots

print("The length of the data corpus is : {}".format(len(preprocessed_movie_plots)))

dataframe.head()
```

100%|██████████| 14781/14781 [03:33<00:00, 69.10it/s]

The length of the data corpus is : 14781

Out[57]:

	index	title	plot_synopsis	tags	split	CleanedPlots
0	0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	set hamburg west germani sever crimin take adv...
1	1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	grader name griffin bing decid gather entir gr...
2	2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	gari hook new recruit british armi take leav m...
3	3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	sergeant dragon jacki chan part hong kong mari...
4	4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	pretoria south africa major charl bolton rod m...

```
In [6]:

#Data cleaning without stemming for use with word vectors

preprocessed_movie_plots =[]

for movie_plot in tqdm(dataframe['plot_synopsis'].values):
    filtered_sentence=[]
    movie_plot = decontracted(movie_plot)
    movie_plot = removeNumbers(movie_plot)
    movie_plot = removeHtml(movie_plot)
    movie_plot = removeURL(movie_plot)
    movie_plot = removePunctuations(movie_plot)
    movie_plot = removePatterns(movie_plot)

    for cleaned_words in movie_plot.split():
        if((cleaned_words not in custom_stopwords) and (len(cleaned_words)>2)):
            word=cleaned_words.lower()
            filtered_sentence.append(word)
        else:
            continue
    movie_plot = " ".join(filtered_sentence) #Final string of cleaned words
    preprocessed_movie_plots.append(movie_plot.strip()) #Data corpus contaning cleaned movie_plots from the whole dataset

#Adding a column of CleanedPlots to the table final which stores the data_corpus after pre-processing the movie_p
lots
dataframe['CleanedPlots_NoStemming']=preprocessed_movie_plots

print("The length of the data corpus is : {}".format(len(preprocessed_movie_plots)))

dataframe.head()
```

100%|██████████| 14781/14781 [00:56<00:00, 259.53it/s]

The length of the data corpus is : 14781

Out[6]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming
0	0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	set hamburg west germani sever crimin take adv...	set hamburg west germany several criminals tak...
1	1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	grader name griffin bing decid gather entir gr...	grader named griffin bing decides gather entir...
2	2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	gari hook new recruit british armi take leav m...	gary hook new recruit british army takes leave...
3	3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	sergeant dragon jacki chan part hong kong mari...	sergeant dragon jackie chan part hong kong mar...
4	4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	pretoria south africa major charl bolton rod m...	pretoria south africa major charles bolton rod...

We will create a new dataset to store the cleaned movie plot synopses

```
In [7]:

dataframe.to_csv("cleaned_movie_plots.csv", index=False)
```

```
In [8]:

print("Number of data points in sample :", dataframe.shape[0])
print("Number of dimensions :", dataframe.shape[1])
```

Number of data points in sample : 14781
Number of dimensions : 7

8. Machine Learning Models with OneVsRest

8.1 Splitting the dataset into train and test

Here, since we will perform random search cross validation, we will take the "train" and "validation" data mentioned in the "split" column as our total training data.

In [3]:

```
#Load the processed dataset
dataframe=pd.read_csv("cleaned_movie_plots.csv")
dataframe.head()
```

Out[3]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming
0	0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	set hamburg west germani sever crimin take adv...	set hamburg west germany several criminals tak...
1	1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	grader name griffin bing decid gather entir gr...	grader named griffin bing decides gather entir...
2	2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	gari hook new recruit british armi take leav m...	gary hook new recruit british army takes leave...
3	3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	sergeant dragon jacki chan part hong kong mari...	sergeant dragon jackie chan part hong kong mar...
4	4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	pretoria south africa major charl bolton rod m...	pretoria south africa major charles bolton rod...

In [4]:

```
#Create a dataset for train and test
data_test=dataframe.loc[(dataframe['split'] == 'test')]
data_train=dataframe.loc[(dataframe['split'] == 'val') | (dataframe['split'] == 'train')]

#Split the whole data into train and test set
X_train = data_train['CleanedPlots']
y_train = data_train['tags']

X_test = data_test['CleanedPlots']
y_test = data_test['tags']

print("Number of points in training data: ",data_train.shape[0])
print("Number of points in test data: ",data_test.shape[0])
```

Number of points in training data: 11816
Number of points in test data: 2965

Convert the tags to binary vectors for multi label classification

X	t1	t2	t3	t4
x1	0	1	1	0
x1	1	0	0	0
x1	0	1	0	0

In [8]:

```
# binary='true' will give a binary vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

In [9]:

```
y_train_multilabel
```

Out[9]:

<11816x71 sparse matrix of type '<class 'numpy.int64'>' with 35129 stored elements in Compressed Sparse Row format>

In [10]:

```
y_test_multilabel
```

Out[10]:

```
<2965x71 sparse matrix of type '<class 'numpy.int64'>'
  with 9021 stored elements in Compressed Sparse Row format>
```

8.2 Featurizing data with TF-IDF vectorizer (1-Grams)

In [10]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:04.556160

In [11]:

```
print("Dimensions of train data X:",X_train_multilabel.shape, "Y :",y_train_multilabel.shape)
print("Dimensions of test data X:",X_test_multilabel.shape,"Y:",y_test_multilabel.shape)
```

Dimensions of train data X: (11816, 39462) Y : (11816, 71)
Dimensions of test data X: (2965, 39462) Y: (2965, 71)

8.2.1 Applying Logistic Regression with OneVsRest Classifier

In [72]:

```
from sklearn.linear_model import LogisticRegression
start = datetime.now()

classifier1 = OneVsRestClassifier(LogisticRegression(penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier1.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier1.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.01821247892074199
Hamming loss 0.08066883594993231

Micro-average quality numbers
Precision: 0.2542, Recall: 0.4562, F1-measure: 0.3264

Macro-average quality numbers
Precision: 0.1250, Recall: 0.2475, F1-measure: 0.1635

Classification Report	precision	recall	f1-score	support
0	0.07	0.14	0.09	56
1	0.19	0.42	0.26	129
2	0.08	0.18	0.11	28
3	0.05	0.09	0.06	22
4	0.19	0.61	0.29	18

5	0.04	0.11	0.06	35
6	0.19	0.40	0.26	30
7	0.06	0.14	0.08	79
8	0.00	0.00	0.00	8
9	0.08	0.18	0.11	45
10	0.12	0.27	0.16	11
11	0.03	0.07	0.04	40
12	0.07	0.15	0.10	115
13	0.07	0.17	0.09	18
14	0.03	0.11	0.05	9
15	0.00	0.00	0.00	15
16	0.00	0.00	0.00	13
17	0.23	0.44	0.31	368
18	0.07	0.15	0.09	27
19	0.08	0.15	0.10	97
20	0.32	0.54	0.40	551
21	0.02	0.05	0.03	41
22	0.08	0.19	0.11	85
23	0.08	0.17	0.10	42
24	0.07	0.17	0.10	83
25	0.13	0.27	0.18	159
26	0.23	0.39	0.29	112
27	0.03	0.09	0.05	11
28	0.30	0.49	0.38	596
29	0.27	0.53	0.36	190
30	0.22	0.52	0.31	83
31	0.06	0.17	0.09	12
32	0.10	0.27	0.14	26
33	0.10	0.19	0.13	64
34	0.10	0.24	0.14	25
35	0.05	0.17	0.08	29
36	0.20	0.52	0.29	92
37	0.11	0.22	0.15	172
38	0.14	0.28	0.19	136
39	0.06	0.09	0.07	32
40	0.05	0.10	0.07	40
41	0.00	0.00	0.00	5
42	0.07	0.17	0.10	93
43	0.65	0.70	0.67	1155
44	0.11	0.23	0.15	117
45	0.19	0.51	0.27	145
46	0.00	0.00	0.00	5
47	0.21	0.42	0.28	129
48	0.05	0.17	0.08	36
49	0.08	0.16	0.11	44
50	0.09	0.25	0.13	28
51	0.18	0.37	0.24	51
52	0.28	0.47	0.35	395
53	0.08	0.17	0.11	65
54	0.02	0.07	0.03	15
55	0.05	0.11	0.07	37
56	0.32	0.55	0.41	507
57	0.42	0.60	0.49	587
58	0.14	0.24	0.18	148
59	0.16	0.30	0.21	173
60	0.18	0.41	0.25	66
61	0.08	0.20	0.12	51
62	0.03	0.09	0.05	66
63	0.03	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.21	0.50	0.29	228
66	0.03	0.07	0.04	27
67	0.08	0.15	0.10	119
68	0.54	0.70	0.61	911
69	0.22	0.43	0.29	14
70	0.00	0.00	0.00	13

micro avg	0.25	0.46	0.33	9021
macro avg	0.12	0.25	0.16	9021
weighted avg	0.30	0.46	0.36	9021
samples avg	0.27	0.50	0.30	9021

Time taken to run this cell : 0:00:18.450013

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

'precision', 'predicted', average, warn_for)

In [73]:

```
joblib.dump(classifier1, 'ovr_with_lr_clf1.pkl')
```

Out[73]:

```
['ovr_with_lr_clf1.pkl']
```

8.2.2 Applying Logistic Regression with OneVsRest Classifier + SGDClassifier

In [74]:

```
start = datetime.now()

classifier2 = OneVsRestClassifier(SGDClassifier(loss='log',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier2.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier2.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.0030354131534569982
Hamming loss 0.13473624207301144

Micro-average quality numbers
Precision: 0.1577, Recall: 0.4937, F1-measure: 0.2390

Macro-average quality numbers
Precision: 0.0936, Recall: 0.3335, F1-measure: 0.1325

Classification Report				
	precision	recall	f1-score	support
0	0.03	0.14	0.05	56
1	0.13	0.43	0.20	129
2	0.04	0.39	0.07	28
3	0.01	0.14	0.02	22
4	0.04	0.61	0.08	18
5	0.04	0.31	0.07	35
6	0.06	0.40	0.10	30
7	0.05	0.25	0.09	79
8	0.01	0.25	0.01	8
9	0.03	0.18	0.06	45
10	0.03	0.64	0.05	11
11	0.03	0.20	0.05	40
12	0.06	0.23	0.10	115
13	0.02	0.28	0.03	18
14	0.01	0.44	0.03	9
15	0.01	0.20	0.02	15
16	0.00	0.08	0.01	13
17	0.22	0.46	0.29	368
18	0.02	0.19	0.04	27
19	0.06	0.25	0.10	97
20	0.32	0.58	0.41	551
21	0.03	0.20	0.05	41
22	0.07	0.33	0.12	85
23	0.03	0.19	0.05	42
24	0.06	0.27	0.10	83
25	0.11	0.35	0.17	159
26	0.19	0.49	0.27	112
27	0.00	0.09	0.01	11
28	0.28	0.55	0.37	596
29	0.25	0.58	0.35	190

30	0.15	0.60	0.24	83
31	0.02	0.42	0.03	12
32	0.05	0.50	0.10	26
33	0.07	0.27	0.11	64
34	0.05	0.36	0.09	25
35	0.02	0.21	0.04	29
36	0.15	0.58	0.24	92
37	0.11	0.30	0.16	172
38	0.10	0.29	0.15	136
39	0.02	0.16	0.03	32
40	0.03	0.25	0.06	40
41	0.00	0.20	0.01	5
42	0.06	0.26	0.10	93
43	0.66	0.70	0.68	1155
44	0.08	0.32	0.13	117
45	0.16	0.54	0.25	145
46	0.00	0.20	0.01	5
47	0.15	0.42	0.22	129
48	0.03	0.22	0.05	36
49	0.03	0.20	0.05	44
50	0.02	0.21	0.04	28
51	0.04	0.22	0.06	51
52	0.29	0.47	0.36	395
53	0.03	0.15	0.06	65
54	0.01	0.27	0.03	15
55	0.03	0.22	0.05	37
56	0.30	0.55	0.39	507
57	0.42	0.60	0.49	587
58	0.11	0.34	0.17	148
59	0.14	0.35	0.20	173
60	0.13	0.55	0.21	66
61	0.04	0.24	0.07	51
62	0.03	0.15	0.05	66
63	0.02	0.21	0.04	39
64	0.00	0.12	0.01	8
65	0.17	0.54	0.26	228
66	0.02	0.26	0.04	27
67	0.05	0.20	0.08	119
68	0.53	0.70	0.61	911
69	0.03	0.57	0.06	14
70	0.00	0.08	0.01	13
micro avg	0.16	0.49	0.24	9021
macro avg	0.09	0.33	0.13	9021
weighted avg	0.28	0.49	0.34	9021
samples avg	0.17	0.53	0.23	9021

Time taken to run this cell : 0:00:04.447970

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.
'precision', 'predicted', average, warn_for)

In [75]:

```
from sklearn.externals import joblib
joblib.dump(classifier2, 'ovr_with_lr_sgd_clf2.pkl')
```

Out[75]:

['ovr_with_lr_sgd_clf2.pkl']

8.2.3 Applying Linear SVM with OneVsRest Classifier + SGDClassifier with 'hinge' loss

In [100]:

```
start = datetime.now()

classifier2 = OneVsRestClassifier(SGDClassifier(loss='hinge',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier2.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier2.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.003372681281618887
Hamming loss 0.1436572215756597

Micro-average quality numbers
Precision: 0.1437, Recall: 0.4744, F1-measure: 0.2206

Macro-average quality numbers
Precision: 0.0897, Recall: 0.3445, F1-measure: 0.1272

Classification Report

	precision	recall	f1-score	support
0	0.04	0.25	0.06	56
1	0.14	0.46	0.21	129
2	0.03	0.29	0.06	28
3	0.02	0.23	0.03	22
4	0.04	0.50	0.07	18
5	0.01	0.14	0.03	35
6	0.05	0.50	0.09	30
7	0.04	0.25	0.07	79
8	0.01	0.25	0.02	8
9	0.04	0.27	0.06	45
10	0.02	0.45	0.03	11
11	0.02	0.17	0.04	40
12	0.05	0.21	0.08	115
13	0.01	0.28	0.03	18
14	0.02	0.67	0.04	9
15	0.01	0.13	0.01	15
16	0.00	0.08	0.01	13
17	0.21	0.39	0.27	368
18	0.03	0.30	0.06	27
19	0.08	0.37	0.13	97
20	0.31	0.50	0.38	551
21	0.03	0.22	0.05	41
22	0.06	0.27	0.09	85
23	0.04	0.29	0.06	42
24	0.07	0.30	0.11	83
25	0.10	0.33	0.15	159
26	0.17	0.53	0.26	112
27	0.01	0.27	0.02	11
28	0.29	0.48	0.36	596
29	0.21	0.57	0.30	190
30	0.12	0.54	0.20	83
31	0.02	0.42	0.03	12
32	0.05	0.46	0.09	26
33	0.07	0.31	0.11	64
34	0.05	0.44	0.09	25
35	0.03	0.34	0.06	29
36	0.12	0.54	0.20	92
37	0.09	0.27	0.13	172
38	0.11	0.35	0.17	136
39	0.03	0.25	0.05	32
40	0.02	0.17	0.04	40
41	0.00	0.00	0.00	5

42	0.06	0.27	0.10	93
43	0.65	0.69	0.67	1155
44	0.08	0.32	0.13	117
45	0.15	0.52	0.23	145
46	0.00	0.20	0.01	5
47	0.14	0.45	0.21	129
48	0.03	0.28	0.05	36
49	0.02	0.18	0.04	44
50	0.04	0.43	0.07	28
51	0.05	0.31	0.09	51
52	0.24	0.45	0.31	395
53	0.04	0.23	0.07	65
54	0.03	0.47	0.05	15
55	0.03	0.22	0.05	37
56	0.30	0.50	0.38	507
57	0.41	0.59	0.48	587
58	0.11	0.33	0.16	148
59	0.13	0.39	0.19	173
60	0.10	0.50	0.16	66
61	0.05	0.33	0.09	51
62	0.02	0.14	0.04	66
63	0.02	0.18	0.04	39
64	0.01	0.25	0.01	8
65	0.17	0.51	0.25	228
66	0.02	0.22	0.04	27
67	0.06	0.23	0.09	119
68	0.54	0.66	0.59	911
69	0.04	0.50	0.07	14
70	0.00	0.08	0.01	13
micro avg	0.14	0.47	0.22	9021
macro avg	0.09	0.34	0.13	9021
weighted avg	0.28	0.47	0.33	9021
samples avg	0.16	0.51	0.21	9021

Time taken to run this cell : 0:00:04.079711

In [101]:

```
from sklearn.externals import joblib
joblib.dump(classifier2, 'ovr_with_svm_sgd_clf2.pkl')
```

Out[101]:

```
['ovr_with_svm_sgd_clf2.pkl']
```

8.3 Featurizing data with Tfidf vectorizer (1-2 Grams)

In [104]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), sublinear_tf=False, ngram_range=(1,2))
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 : 0:00:29.360189

In [105]:

```
print("Dimensions of train data X:",X_train_multilabel.shape, "Y :",y_train_multilabel.shape)
print("Dimensions of test data X:",X_test_multilabel.shape,"Y:",y_test_multilabel.shape)
```

```
Dimensions of train data X: (11816, 718424) Y : (11816, 71)
Dimensions of test data X: (2965, 718424) Y: (2965, 71)
```

8.3.1 Applying Logistic Regression with OneVsRest Classifier

In [82]:

```
start = datetime.now()

classifier3 = OneVsRestClassifier(LogisticRegression(penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier3.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier3.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.017537942664418212
Hamming loss 0.08845925468493931

Micro-average quality numbers
Precision: 0.2387, Recall: 0.4863, F1-measure: 0.3203

Macro-average quality numbers
Precision: 0.1207, Recall: 0.2776, F1-measure: 0.1647

Classification Report

	precision	recall	f1-score	support
0	0.06	0.16	0.09	56
1	0.18	0.47	0.25	129
2	0.07	0.18	0.10	28
3	0.04	0.09	0.06	22
4	0.17	0.61	0.27	18
5	0.05	0.17	0.07	35
6	0.12	0.33	0.18	30
7	0.05	0.16	0.08	79
8	0.00	0.00	0.00	8
9	0.10	0.27	0.15	45
10	0.14	0.36	0.21	11
11	0.03	0.10	0.05	40
12	0.07	0.17	0.10	115
13	0.05	0.17	0.08	18
14	0.03	0.11	0.05	9
15	0.02	0.07	0.03	15
16	0.02	0.08	0.03	13
17	0.22	0.48	0.30	368
18	0.06	0.15	0.08	27
19	0.08	0.22	0.12	97
20	0.32	0.60	0.41	551
21	0.01	0.02	0.01	41
22	0.09	0.24	0.13	85
23	0.10	0.24	0.14	42
24	0.08	0.25	0.12	83
25	0.13	0.31	0.19	159
26	0.23	0.45	0.31	112
27	0.03	0.09	0.04	11
28	0.31	0.52	0.39	596
29	0.26	0.56	0.35	190
30	0.19	0.54	0.28	83
31	0.06	0.17	0.09	12
32	0.10	0.35	0.16	26
33	0.09	0.22	0.13	64
34	0.09	0.24	0.13	25
35	0.04	0.14	0.06	29
36	0.18	0.58	0.28	92
37	0.12	0.29	0.17	172
38	0.15	0.36	0.21	136
39	0.05	0.09	0.07	32
40	0.04	0.10	0.06	40
41	0.00	0.00	0.00	5

42	0.10	0.28	0.14	93
43	0.65	0.71	0.68	1155
44	0.10	0.26	0.15	117
45	0.17	0.57	0.27	145
46	0.00	0.00	0.00	5
47	0.20	0.47	0.28	129
48	0.05	0.19	0.08	36
49	0.07	0.16	0.10	44
50	0.08	0.29	0.12	28
51	0.14	0.35	0.20	51
52	0.28	0.50	0.36	395
53	0.07	0.18	0.10	65
54	0.06	0.27	0.09	15
55	0.04	0.11	0.06	37
56	0.32	0.59	0.41	507
57	0.41	0.61	0.49	587
58	0.15	0.32	0.20	148
59	0.15	0.34	0.21	173
60	0.16	0.42	0.23	66
61	0.07	0.18	0.10	51
62	0.03	0.12	0.05	66
63	0.02	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.19	0.54	0.28	228
66	0.04	0.11	0.06	27
67	0.07	0.15	0.09	119
68	0.53	0.70	0.61	911
69	0.18	0.50	0.26	14
70	0.00	0.00	0.00	13
micro avg	0.24	0.49	0.32	9021
macro avg	0.12	0.28	0.16	9021
weighted avg	0.30	0.49	0.36	9021
samples avg	0.26	0.52	0.30	9021

Time taken to run this cell : 0:01:46.821897

In [83]:

```
joblib.dump(classifier3, 'ovr_with_lr_clf3_bigrams.pkl')
```

Out[83]:

```
['ovr_with_lr_clf3_bigrams.pkl']
```

8.3.2 Applying Logistic Regression with OneVsRest Classifier + SGDClassifier

In [85]:

```
start = datetime.now()

classifier4 = OneVsRestClassifier(SGDClassifier(loss='log',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier4.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier4.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.004721753794266442
Hamming loss  0.136170819181531
```

```
Micro-average quality numbers
Precision: 0.1608, Recall: 0.5160, F1-measure: 0.2452
```


Macro-average quality numbers
Precision: 0.0952, Recall: 0.3537, F1-measure: 0.1371

Classification Report				
	precision	recall	f1-score	support
0	0.02	0.14	0.04	56
1	0.17	0.49	0.25	129
2	0.03	0.29	0.05	28
3	0.01	0.09	0.02	22
4	0.04	0.56	0.08	18
5	0.03	0.31	0.06	35
6	0.06	0.53	0.11	30
7	0.05	0.27	0.09	79
8	0.01	0.25	0.01	8
9	0.04	0.27	0.07	45
10	0.02	0.36	0.04	11
11	0.02	0.15	0.03	40
12	0.06	0.23	0.09	115
13	0.02	0.39	0.04	18
14	0.02	0.44	0.04	9
15	0.01	0.27	0.02	15
16	0.01	0.15	0.01	13
17	0.21	0.41	0.28	368
18	0.03	0.19	0.04	27
19	0.06	0.23	0.09	97
20	0.31	0.58	0.40	551
21	0.06	0.37	0.10	41
22	0.06	0.29	0.10	85
23	0.03	0.19	0.05	42
24	0.06	0.28	0.10	83
25	0.12	0.40	0.19	159
26	0.18	0.46	0.25	112
27	0.01	0.18	0.02	11
28	0.28	0.57	0.38	596
29	0.24	0.62	0.34	190
30	0.14	0.60	0.23	83
31	0.01	0.17	0.01	12
32	0.06	0.50	0.10	26
33	0.07	0.34	0.12	64
34	0.03	0.28	0.06	25
35	0.02	0.14	0.03	29
36	0.14	0.59	0.23	92
37	0.10	0.32	0.15	172
38	0.12	0.38	0.18	136
39	0.03	0.28	0.06	32
40	0.04	0.30	0.07	40
41	0.00	0.00	0.00	5
42	0.06	0.26	0.10	93
43	0.65	0.71	0.68	1155
44	0.09	0.32	0.14	117
45	0.17	0.66	0.27	145
46	0.00	0.20	0.01	5
47	0.16	0.50	0.24	129
48	0.04	0.25	0.06	36
49	0.03	0.23	0.06	44
50	0.04	0.39	0.07	28
51	0.06	0.35	0.11	51
52	0.28	0.53	0.37	395
53	0.04	0.20	0.06	65
54	0.02	0.33	0.04	15
55	0.04	0.32	0.08	37
56	0.30	0.59	0.40	507
57	0.43	0.61	0.50	587
58	0.11	0.37	0.17	148
59	0.13	0.36	0.19	173
60	0.12	0.58	0.20	66
61	0.04	0.25	0.07	51
62	0.03	0.20	0.05	66
63	0.03	0.26	0.05	39
64	0.01	0.25	0.01	8
65	0.18	0.57	0.27	228
66	0.02	0.22	0.04	27
67	0.06	0.20	0.10	119
68	0.52	0.73	0.61	911
69	0.06	0.64	0.11	14
70	0.01	0.15	0.02	13
micro avg	0.16	0.52	0.25	9021
macro avg	0.10	0.35	0.14	9021
weighted avg	0.28	0.52	0.35	9021
samples avg	0.17	0.55	0.23	9021

Time taken to run this cell : 0:00:30.436438

In [88]:

```
joblib.dump(classifier4, 'ovr_with_sgd_clf4_bigrams.pkl')
```

Out[88]:

```
['ovr_with_sgd_clf4_bigrams.pkl']
```

8.3.3 Applying Linear SVM with OneVsRest Classifier + SGDClassifier with 'hinge' loss

In [106]:

```
start = datetime.now()

classifier3 = OneVsRestClassifier(SGDClassifier(loss='hinge',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier3.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier3.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.0030354131534569982
Hamming loss 0.14862598864688978

Micro-average quality numbers
Precision: 0.1476, Recall: 0.5169, F1-measure: 0.2296

Macro-average quality numbers
Precision: 0.0916, Recall: 0.3802, F1-measure: 0.1326

Classification Report				
	precision	recall	f1-score	support
0	0.04	0.27	0.07	56
1	0.14	0.55	0.23	129
2	0.03	0.32	0.06	28
3	0.01	0.18	0.02	22
4	0.04	0.72	0.08	18
5	0.05	0.46	0.09	35
6	0.05	0.50	0.09	30
7	0.06	0.32	0.10	79
8	0.01	0.62	0.03	8
9	0.05	0.33	0.08	45
10	0.02	0.36	0.04	11
11	0.03	0.20	0.05	40
12	0.05	0.27	0.09	115
13	0.01	0.17	0.02	18
14	0.02	0.56	0.03	9
15	0.02	0.33	0.04	15
16	0.01	0.15	0.01	13
17	0.21	0.45	0.28	368
18	0.03	0.26	0.05	27
19	0.07	0.34	0.11	97
20	0.33	0.52	0.41	551
21	0.03	0.22	0.05	41
22	0.07	0.39	0.12	85
23	0.02	0.12	0.03	42
24	0.06	0.31	0.10	83
25	0.11	0.37	0.16	159
26	0.14	0.46	0.22	112

27	0.01	0.18	0.01	11
28	0.30	0.53	0.39	596
29	0.21	0.64	0.32	190
30	0.14	0.61	0.22	83
31	0.01	0.25	0.02	12
32	0.04	0.42	0.07	26
33	0.09	0.44	0.14	64
34	0.04	0.36	0.07	25
35	0.02	0.21	0.03	29
36	0.13	0.63	0.21	92
37	0.10	0.40	0.16	172
38	0.10	0.38	0.16	136
39	0.03	0.28	0.05	32
40	0.03	0.30	0.06	40
41	0.00	0.00	0.00	5
42	0.08	0.37	0.13	93
43	0.65	0.72	0.68	1155
44	0.07	0.30	0.12	117
45	0.14	0.59	0.23	145
46	0.01	0.40	0.02	5
47	0.14	0.48	0.22	129
48	0.02	0.17	0.03	36
49	0.02	0.20	0.04	44
50	0.04	0.32	0.06	28
51	0.05	0.33	0.09	51
52	0.26	0.47	0.34	395
53	0.05	0.28	0.09	65
54	0.02	0.40	0.04	15
55	0.03	0.27	0.05	37
56	0.30	0.52	0.38	507
57	0.40	0.62	0.49	587
58	0.10	0.32	0.16	148
59	0.12	0.43	0.19	173
60	0.10	0.53	0.17	66
61	0.05	0.31	0.08	51
62	0.03	0.18	0.05	66
63	0.03	0.28	0.06	39
64	0.01	0.38	0.02	8
65	0.17	0.57	0.26	228
66	0.03	0.37	0.05	27
67	0.06	0.25	0.10	119
68	0.53	0.74	0.62	911
69	0.04	0.57	0.07	14
70	0.01	0.23	0.02	13

micro avg	0.15	0.52	0.23	9021
macro avg	0.09	0.38	0.13	9021
weighted avg	0.28	0.52	0.34	9021
samples avg	0.16	0.55	0.22	9021

Time taken to run this cell : 0:00:25.127653

In [107]:

```
from sklearn.externals import joblib
joblib.dump(classifier3, 'ovr_with_svm_sgd_clf3.pkl')
```

Out[107]:

```
['ovr_with_svm_sgd_clf3.pkl']
```

8.4 Featurizing data with Tfidf vectorizer (1-3 Grams)

In [108]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:01:03.432895

In [109]:

```
print("Dimensions of train data X:",X_train_multilabel.shape, "Y :",y_train_multilabel.shape)
print("Dimensions of test data X:",X_test_multilabel.shape,"Y:",y_test_multilabel.shape)
```

```
Dimensions of train data X: (11816, 100000) Y : (11816, 71)
Dimensions of test data X: (2965, 100000) Y: (2965, 71)
```

8.4.1 Applying Logistic Regression with OneVsRest Classifier

In [91]:

```
start = datetime.now()

classifier5 = OneVsRestClassifier(LogisticRegression(penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier5.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier5.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.017200674536256323
Hamming loss 0.08469705246657008
```

```
Micro-average quality numbers
Precision: 0.2457, Recall: 0.4719, F1-measure: 0.3232
```

```
Macro-average quality numbers
Precision: 0.1222, Recall: 0.2618, F1-measure: 0.1636
```

```
Classification Report
      precision    recall  f1-score   support

0         0.06        0.14        0.08         56
1         0.18        0.45        0.26        129
2         0.07        0.18        0.10         28
3         0.04        0.09        0.06         22
4         0.18        0.61        0.27         18
5         0.05        0.14        0.07         35
6         0.15        0.37        0.22         30
7         0.07        0.19        0.10         79
8         0.00        0.00        0.00          8
9         0.08        0.18        0.11         45
10        0.15        0.36        0.21         11
11        0.03        0.07        0.04         40
12        0.07        0.17        0.10        115
13        0.05        0.17        0.08         18
14        0.03        0.11        0.05          9
15        0.02        0.07        0.03         15
16        0.00        0.00        0.00         13
17        0.22        0.46        0.30        368
18        0.08        0.19        0.11         27
19        0.08        0.18        0.11         97
20        0.32        0.58        0.41        551
21        0.01        0.02        0.02         41
22        0.08        0.22        0.12         85
23        0.11        0.24        0.15         42
24        0.08        0.23        0.12         83
25        0.13        0.30        0.18        159
26        0.21        0.39        0.28        112
27        0.03        0.09        0.04         11
28        0.30        0.50        0.38        596
29        0.26        0.55        0.35        190
30        0.20        0.51        0.28         83
```

31	0.06	0.17	0.09	12
32	0.11	0.35	0.17	26
33	0.11	0.23	0.15	64
34	0.09	0.24	0.13	25
35	0.05	0.17	0.08	29
36	0.18	0.52	0.27	92
37	0.12	0.25	0.16	172
38	0.15	0.31	0.20	136
39	0.05	0.09	0.07	32
40	0.05	0.10	0.06	40
41	0.00	0.00	0.00	5
42	0.09	0.23	0.13	93
43	0.65	0.70	0.67	1155
44	0.10	0.23	0.14	117
45	0.18	0.54	0.27	145
46	0.00	0.00	0.00	5
47	0.19	0.43	0.27	129
48	0.05	0.19	0.08	36
49	0.08	0.18	0.11	44
50	0.08	0.25	0.12	28
51	0.14	0.33	0.20	51
52	0.28	0.48	0.35	395
53	0.07	0.15	0.09	65
54	0.02	0.07	0.03	15
55	0.05	0.11	0.06	37
56	0.32	0.58	0.41	507
57	0.40	0.61	0.49	587
58	0.16	0.32	0.21	148
59	0.18	0.35	0.23	173
60	0.17	0.42	0.25	66
61	0.07	0.18	0.10	51
62	0.04	0.12	0.06	66
63	0.03	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.19	0.52	0.28	228
66	0.03	0.07	0.04	27
67	0.06	0.13	0.08	119
68	0.54	0.71	0.61	911
69	0.19	0.43	0.27	14
70	0.00	0.00	0.00	13
micro avg	0.25	0.47	0.32	9021
macro avg	0.12	0.26	0.16	9021
weighted avg	0.30	0.47	0.36	9021
samples avg	0.26	0.51	0.30	9021

Time taken to run this cell : 0:00:28.079117

In [92]:

```
joblib.dump(classifier5, 'ovr_with_lr_clf5_3ngrams.pkl')
```

Out[92]:

```
['ovr_with_lr_clf5_3ngrams.pkl']
```

8.4.2 Applying Linear SVM with OneVsRest Classifier + SGDClassifier with 'hinge' loss

In [110]:

```
start = datetime.now()

classifier5 = OneVsRestClassifier(SGDClassifier(loss='hinge',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier5.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier5.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.002360876897133221
Hamming loss 0.14654537681400376

Micro-average quality numbers
Precision: 0.1465, Recall: 0.5014, F1-measure: 0.2267

Macro-average quality numbers
Precision: 0.0908, Recall: 0.3610, F1-measure: 0.1303

Classification Report				
	precision	recall	f1-score	support
0	0.04	0.25	0.07	56
1	0.12	0.43	0.18	129
2	0.03	0.29	0.05	28
3	0.01	0.18	0.03	22
4	0.04	0.67	0.08	18
5	0.02	0.20	0.03	35
6	0.05	0.43	0.08	30
7	0.05	0.28	0.09	79
8	0.01	0.25	0.01	8
9	0.04	0.31	0.07	45
10	0.02	0.45	0.03	11
11	0.01	0.12	0.03	40
12	0.06	0.29	0.11	115
13	0.01	0.22	0.02	18
14	0.03	0.89	0.05	9
15	0.01	0.20	0.01	15
16	0.01	0.15	0.01	13
17	0.21	0.44	0.28	368
18	0.02	0.19	0.04	27
19	0.06	0.28	0.10	97
20	0.30	0.55	0.39	551
21	0.03	0.24	0.05	41
22	0.06	0.29	0.10	85
23	0.03	0.24	0.06	42
24	0.05	0.27	0.08	83
25	0.12	0.42	0.19	159
26	0.15	0.45	0.22	112
27	0.01	0.18	0.01	11
28	0.31	0.44	0.36	596
29	0.21	0.58	0.31	190
30	0.13	0.57	0.22	83
31	0.02	0.42	0.04	12
32	0.04	0.42	0.07	26
33	0.05	0.30	0.09	64
34	0.05	0.48	0.09	25
35	0.03	0.28	0.05	29
36	0.12	0.55	0.19	92
37	0.11	0.33	0.16	172
38	0.10	0.35	0.16	136
39	0.04	0.38	0.07	32
40	0.03	0.20	0.05	40
41	0.00	0.00	0.00	5

42	0.08	0.35	0.14	93
43	0.65	0.70	0.68	1155
44	0.07	0.32	0.12	117
45	0.14	0.61	0.23	145
46	0.00	0.20	0.01	5
47	0.14	0.47	0.22	129
48	0.03	0.19	0.05	36
49	0.03	0.23	0.05	44
50	0.04	0.32	0.06	28
51	0.05	0.31	0.09	51
52	0.26	0.52	0.35	395
53	0.05	0.23	0.08	65
54	0.01	0.27	0.02	15
55	0.02	0.22	0.04	37
56	0.29	0.56	0.38	507
57	0.40	0.62	0.49	587
58	0.12	0.41	0.18	148
59	0.13	0.36	0.19	173
60	0.14	0.65	0.23	66
61	0.05	0.35	0.08	51
62	0.03	0.18	0.05	66
63	0.03	0.28	0.05	39
64	0.01	0.38	0.02	8
65	0.17	0.56	0.26	228
66	0.02	0.22	0.04	27
67	0.06	0.24	0.10	119
68	0.54	0.70	0.61	911
69	0.05	0.71	0.09	14
70	0.00	0.00	0.00	13
micro avg	0.15	0.50	0.23	9021
macro avg	0.09	0.36	0.13	9021
weighted avg	0.28	0.50	0.34	9021
samples avg	0.15	0.53	0.21	9021

Time taken to run this cell : 0:00:06.888086

In [111]:

```
from sklearn.externals import joblib
joblib.dump(classifier5, 'ovr_with_svm_sgd_clf5.pkl')
```

Out[111]:

```
['ovr_with_svm_sgd_clf5.pkl']
```

8.5 Featurizing data with Tfidf vectorizer (1-4 Grams)

In [112]:

```
start = datetime.now()
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,4))
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 : 0:01:39.370960

In [113]:

```
print("Dimensions of train data X:",X_train_multilabel.shape, "Y :",y_train_multilabel.shape)
print("Dimensions of test data X:",X_test_multilabel.shape,"Y:",y_test_multilabel.shape)
```

```
Dimensions of train data X: (11816, 100000) Y : (11816, 71)
Dimensions of test data X: (2965, 100000) Y: (2965, 71)
```

8.5.1 Applying Logistic Regression with OneVsRest Classifier

In [95]:

```
start = datetime.now()

classifier6 = OneVsRestClassifier(LogisticRegression(penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier6.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier6.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

Accuracy : 0.01821247892074199
Hamming loss 0.0846020473600456

Micro-average quality numbers
Precision: 0.2462, Recall: 0.4725, F1-measure: 0.3237

Macro-average quality numbers
Precision: 0.1229, Recall: 0.2630, F1-measure: 0.1645

Classification Report	precision	recall	f1-score	support
0	0.06	0.14	0.08	56
1	0.18	0.45	0.26	129
2	0.07	0.18	0.10	28
3	0.05	0.09	0.06	22
4	0.18	0.61	0.27	18
5	0.05	0.14	0.07	35
6	0.15	0.37	0.21	30
7	0.07	0.19	0.10	79
8	0.00	0.00	0.00	8
9	0.08	0.18	0.11	45
10	0.15	0.36	0.21	11
11	0.03	0.07	0.04	40
12	0.07	0.17	0.10	115
13	0.05	0.17	0.08	18
14	0.03	0.11	0.05	9
15	0.02	0.07	0.03	15
16	0.00	0.00	0.00	13
17	0.22	0.46	0.30	368
18	0.08	0.19	0.11	27
19	0.09	0.20	0.12	97
20	0.32	0.58	0.41	551
21	0.01	0.02	0.02	41
22	0.09	0.24	0.13	85
23	0.11	0.24	0.15	42
24	0.08	0.23	0.12	83
25	0.13	0.30	0.19	159
26	0.21	0.39	0.28	112
27	0.03	0.09	0.04	11
28	0.30	0.50	0.38	596
29	0.26	0.55	0.35	190
30	0.20	0.51	0.28	83
31	0.06	0.17	0.09	12
32	0.11	0.35	0.17	26
33	0.11	0.23	0.15	64
34	0.09	0.24	0.13	25
35	0.05	0.17	0.08	29
36	0.18	0.52	0.27	92
37	0.12	0.25	0.16	172
38	0.15	0.31	0.20	136
39	0.07	0.12	0.09	32
40	0.05	0.10	0.06	40
41	0.00	0.00	0.00	5

42	0.09	0.23	0.13	93
43	0.65	0.70	0.67	1155
44	0.10	0.23	0.14	117
45	0.18	0.54	0.27	145
46	0.00	0.00	0.00	5
47	0.20	0.44	0.28	129
48	0.05	0.19	0.08	36
49	0.08	0.18	0.11	44
50	0.08	0.25	0.12	28
51	0.14	0.33	0.20	51
52	0.28	0.48	0.35	395
53	0.07	0.15	0.09	65
54	0.02	0.07	0.03	15
55	0.04	0.11	0.06	37
56	0.32	0.58	0.41	507
57	0.40	0.61	0.49	587
58	0.15	0.32	0.21	148
59	0.18	0.35	0.23	173
60	0.17	0.42	0.25	66
61	0.07	0.18	0.10	51
62	0.04	0.14	0.06	66
63	0.03	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.19	0.52	0.28	228
66	0.03	0.07	0.04	27
67	0.06	0.13	0.08	119
68	0.54	0.71	0.61	911
69	0.19	0.43	0.27	14
70	0.00	0.00	0.00	13
micro avg	0.25	0.47	0.32	9021
macro avg	0.12	0.26	0.16	9021
weighted avg	0.30	0.47	0.36	9021
samples avg	0.26	0.51	0.30	9021

Time taken to run this cell : 0:00:29.418445

In [96]:

```
joblib.dump(classifier6, 'ovr_with_lr_clf6_4ngrams.pkl')
```

Out[96]:

```
['ovr_with_lr_clf6_4ngrams.pkl']
```

8.5.2 Applying Linear SVM with OneVsRest Classifier + SGDClassifier with 'hinge' loss

In [114]:

```
start = datetime.now()

classifier6 = OneVsRestClassifier(SGDClassifier(loss='hinge',penalty='l1', class_weight='balanced'), n_jobs=-1)
classifier6.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier6.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Accuracy : 0.003372681281618887
Hamming loss  0.14492553974776146
```

```
Micro-average quality numbers
Precision: 0.1466, Recall: 0.4938, F1-measure: 0.2260
```

Macro-average quality numbers
Precision: 0.0910, Recall: 0.3605, F1-measure: 0.1301

Classification Report				
	precision	recall	f1-score	support
0	0.02	0.14	0.04	56
1	0.15	0.53	0.24	129
2	0.04	0.36	0.06	28
3	0.02	0.23	0.03	22
4	0.03	0.61	0.06	18
5	0.02	0.23	0.04	35
6	0.06	0.57	0.10	30
7	0.05	0.25	0.08	79
8	0.01	0.38	0.01	8
9	0.03	0.20	0.05	45
10	0.02	0.45	0.04	11
11	0.03	0.28	0.06	40
12	0.06	0.24	0.09	115
13	0.03	0.44	0.06	18
14	0.02	0.78	0.04	9
15	0.01	0.27	0.02	15
16	0.01	0.15	0.01	13
17	0.22	0.46	0.29	368
18	0.03	0.30	0.05	27
19	0.06	0.23	0.09	97
20	0.32	0.49	0.39	551
21	0.03	0.20	0.05	41
22	0.07	0.39	0.12	85
23	0.03	0.21	0.05	42
24	0.07	0.34	0.11	83
25	0.12	0.39	0.18	159
26	0.15	0.45	0.22	112
27	0.01	0.18	0.01	11
28	0.31	0.44	0.36	596
29	0.20	0.58	0.30	190
30	0.12	0.58	0.20	83
31	0.01	0.25	0.02	12
32	0.04	0.50	0.08	26
33	0.07	0.31	0.11	64
34	0.04	0.40	0.08	25
35	0.03	0.28	0.05	29
36	0.14	0.67	0.23	92
37	0.10	0.33	0.15	172
38	0.11	0.35	0.16	136
39	0.02	0.16	0.03	32
40	0.03	0.25	0.05	40
41	0.00	0.00	0.00	5
42	0.05	0.27	0.09	93
43	0.65	0.72	0.68	1155
44	0.08	0.31	0.12	117
45	0.14	0.52	0.22	145
46	0.00	0.20	0.01	5
47	0.13	0.46	0.21	129
48	0.02	0.17	0.03	36
49	0.03	0.27	0.06	44
50	0.05	0.54	0.10	28
51	0.05	0.27	0.08	51
52	0.25	0.47	0.33	395
53	0.04	0.23	0.07	65
54	0.01	0.20	0.02	15
55	0.02	0.14	0.03	37
56	0.30	0.52	0.38	507
57	0.39	0.61	0.48	587
58	0.10	0.33	0.16	148
59	0.14	0.42	0.21	173
60	0.11	0.61	0.19	66
61	0.05	0.31	0.08	51
62	0.04	0.29	0.08	66
63	0.02	0.21	0.04	39
64	0.01	0.25	0.01	8
65	0.17	0.57	0.26	228
66	0.03	0.26	0.05	27
67	0.05	0.20	0.08	119
68	0.54	0.69	0.61	911
69	0.04	0.57	0.07	14
70	0.01	0.15	0.01	13
micro avg	0.15	0.49	0.23	9021
macro avg	0.09	0.36	0.13	9021
weighted avg	0.28	0.49	0.34	9021
samples avg	0.16	0.53	0.22	9021

Time taken to run this cell : 0:00:07.537110

In [115]:

```
from sklearn.externals import joblib
joblib.dump(classifier6, 'ovr_with_svm_sgd_clf6.pkl')
```

Out[115]:

```
['ovr_with_svm_sgd_clf6.pkl']
```

Observation:

As we can see till now, the LogisticRegression Classifier gave us a much better Micro F1 score than the rest of the classifiers. So, we will proceed to hyperparameter tune the classifiers now to see if we can obtain a better F1 score.

9. Hyperparameter tuning section with Logistic Regression and OneVsRest

Because, we have observed that Logistic Regression gave us a better Micro-F1 score than SVMs

9.1.1 TFIDF with (1-1 Grams)

In [11]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm='l2', tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,1))
X_train_multilabel = vectorizer.fit_transform(X_train)
X_test_multilabel = vectorizer.transform(X_test)

#Convert the tags to binary vectors using sklearn's count vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)

print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:00:04.839401

9.1.2 Get best estimator using RandomSearch + Logistic Regression

In [12]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
l_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```

/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning

```

```

Time taken to perform hyperparameter tuning: 0:52:50.367842
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual
=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.3177680376911081

```

9.1.3 Fit the best estimator on the data

In [13]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, 'lr_ovr_tfidf_hyp_tuned_lgram.pkl')
```

Accuracy : 0.01821247892074199
Hamming loss 0.08066408569460609

Micro-average quality numbers
Precision: 0.2542, Recall: 0.4562, F1-measure: 0.3264

Macro-average quality numbers
Precision: 0.1250, Recall: 0.2475, F1-measure: 0.1635

	precision	recall	f1-score	support
0	0.07	0.14	0.09	56
1	0.19	0.42	0.26	129
2	0.08	0.18	0.11	28
3	0.05	0.09	0.06	22
4	0.19	0.61	0.29	18
5	0.04	0.11	0.06	35
6	0.19	0.40	0.26	30
7	0.06	0.14	0.08	79
8	0.00	0.00	0.00	8
9	0.08	0.18	0.11	45
10	0.12	0.27	0.16	11
11	0.03	0.07	0.04	40
12	0.07	0.15	0.10	115
13	0.07	0.17	0.09	18
14	0.03	0.11	0.05	9
15	0.00	0.00	0.00	15
16	0.00	0.00	0.00	13
17	0.23	0.44	0.31	368
18	0.07	0.15	0.09	27
19	0.08	0.15	0.10	97
20	0.32	0.54	0.40	551
21	0.02	0.05	0.03	41
22	0.08	0.19	0.11	85
23	0.08	0.17	0.10	42
24	0.07	0.17	0.10	83
25	0.13	0.27	0.18	159
26	0.23	0.39	0.29	112
27	0.03	0.09	0.05	11
28	0.30	0.49	0.38	596
29	0.27	0.53	0.36	190
30	0.22	0.52	0.31	83
31	0.06	0.17	0.09	12
32	0.10	0.27	0.14	26
33	0.10	0.19	0.13	64
34	0.10	0.24	0.14	25
35	0.05	0.17	0.08	29
36	0.20	0.52	0.29	92
37	0.11	0.22	0.15	172
38	0.14	0.28	0.19	136

39	0.06	0.09	0.07	32
40	0.05	0.10	0.07	40
41	0.00	0.00	0.00	5
42	0.07	0.17	0.10	93
43	0.65	0.70	0.67	1155
44	0.11	0.23	0.15	117
45	0.19	0.51	0.27	145
46	0.00	0.00	0.00	5
47	0.21	0.42	0.28	129
48	0.05	0.17	0.08	36
49	0.08	0.16	0.11	44
50	0.09	0.25	0.13	28
51	0.18	0.37	0.24	51
52	0.28	0.47	0.35	395
53	0.08	0.17	0.11	65
54	0.02	0.07	0.03	15
55	0.05	0.11	0.07	37
56	0.32	0.55	0.41	507
57	0.42	0.60	0.49	587
58	0.14	0.24	0.18	148
59	0.16	0.30	0.21	173
60	0.18	0.41	0.25	66
61	0.08	0.20	0.12	51
62	0.03	0.09	0.05	66
63	0.03	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.21	0.50	0.29	228
66	0.03	0.07	0.04	27
67	0.08	0.15	0.10	119
68	0.54	0.70	0.61	911
69	0.22	0.43	0.29	14
70	0.00	0.00	0.00	13
micro avg	0.25	0.46	0.33	9021
macro avg	0.12	0.25	0.16	9021
weighted avg	0.30	0.46	0.36	9021
samples avg	0.27	0.50	0.30	9021

Time taken to run this cell : 0:00:18.853082

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

'precision', 'predicted', average, warn_for)

Out[13]:

['lr_ovr_tfidf_hyp_tuned_lgram.pkl']

9.2.1 TFIDF with (1-2 Grams)

In [14]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,2))
X_train_multilabel = vectorizer.fit_transform(X_train)
X_test_multilabel = vectorizer.transform(X_test)

#Convert the tags to binary vectors using sklearn's count vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)

print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:00:28.094092

9.2.2 Get best estimator using RandomSearch + Logistic Regression

In [16]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```

/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
    "timeout or by a memory leak.", UserWarning

```

```

Time taken to perform hyperparameter tuning: 0:36:31.161965
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual
=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.35738148298544803

```

9.2.3 Fit the best estimator on the data

In [17]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, 'lr_ovr_tfidf_hyp_tuned_1_2.pkl')
```

Accuracy : 0.044856661045531196
Hamming loss 0.06280312566800465

Micro-average quality numbers
Precision: 0.3220, Recall: 0.4212, F1-measure: 0.3650

Macro-average quality numbers
Precision: 0.1672, Recall: 0.2130, F1-measure: 0.1833

Classification Report

	precision	recall	f1-score	support
0	0.12	0.09	0.10	56
1	0.20	0.39	0.27	129
2	0.19	0.18	0.18	28
3	0.16	0.14	0.15	22
4	0.32	0.50	0.39	18
5	0.08	0.09	0.08	35
6	0.26	0.30	0.28	30
7	0.10	0.15	0.12	79
8	0.00	0.00	0.00	8
9	0.10	0.09	0.09	45
10	0.30	0.27	0.29	11
11	0.03	0.03	0.03	40
12	0.10	0.13	0.11	115
13	0.06	0.06	0.06	18
14	0.00	0.00	0.00	9
15	0.10	0.07	0.08	15
16	0.00	0.00	0.00	13
17	0.26	0.39	0.31	368
18	0.16	0.19	0.17	27
19	0.11	0.11	0.11	97
20	0.34	0.49	0.40	551
21	0.04	0.05	0.05	41
22	0.12	0.14	0.13	85
23	0.15	0.10	0.12	42
24	0.08	0.12	0.10	83
25	0.20	0.29	0.23	159
26	0.41	0.42	0.42	112
27	0.00	0.00	0.00	11
28	0.30	0.41	0.34	596
29	0.34	0.55	0.42	190
30	0.33	0.59	0.42	83
31	0.15	0.17	0.16	12
32	0.22	0.31	0.25	26
33	0.16	0.20	0.18	64
34	0.18	0.28	0.22	25
35	0.05	0.03	0.04	29
36	0.26	0.53	0.35	92
37	0.19	0.26	0.22	172
38	0.19	0.24	0.21	136
39	0.25	0.09	0.14	32

40	0.11	0.07	0.09	40
41	0.00	0.00	0.00	5
42	0.09	0.13	0.11	93
43	0.64	0.69	0.66	1155
44	0.11	0.17	0.14	117
45	0.23	0.52	0.32	145
46	0.00	0.00	0.00	5
47	0.25	0.36	0.29	129
48	0.10	0.11	0.11	36
49	0.14	0.14	0.14	44
50	0.11	0.07	0.09	28
51	0.16	0.16	0.16	51
52	0.34	0.43	0.38	395
53	0.13	0.14	0.14	65
54	0.00	0.00	0.00	15
55	0.03	0.03	0.03	37
56	0.32	0.46	0.38	507
57	0.43	0.57	0.49	587
58	0.21	0.25	0.23	148
59	0.22	0.25	0.23	173
60	0.27	0.48	0.35	66
61	0.12	0.12	0.12	51
62	0.04	0.05	0.04	66
63	0.04	0.05	0.04	39
64	0.00	0.00	0.00	8
65	0.22	0.43	0.29	228
66	0.07	0.04	0.05	27
67	0.10	0.11	0.11	119
68	0.55	0.67	0.60	911
69	0.23	0.21	0.22	14
70	0.00	0.00	0.00	13
micro avg	0.32	0.42	0.37	9021
macro avg	0.17	0.21	0.18	9021
weighted avg	0.33	0.42	0.36	9021
samples avg	0.34	0.46	0.34	9021

Time taken to run this cell : 0:00:37.607761

```
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.
  'precision', 'predicted', average, warn_for)
```

Out[17]:

```
['lr_ovr_tfidf_hyp_tuned_1_2.pkl']
```

9.3.1 TFIDF with (1-3 Grams)

In [10]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=50000, smooth_idf=True, norm="l2", tokenizer = lambda x
: x.split(" "), sublinear_tf=False, ngram_range=(1,3))
X_train_multilabel = vectorizer.fit_transform(X_train)
X_test_multilabel = vectorizer.transform(X_test)

#Convert the tags to binary vectors using sklearn's count vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)

print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:01:02.372571

9.3.2 Get best estimator using RandomSearch + Logistic Regression

In [11]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 0:28:20.873867
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
fit_intercept=True, intercept_scaling=1, max_iter=100,
multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
solver='warn', tol=0.0001, verbose=0, warm_start=False),
n_jobs=-1)
Best Cross Validation Score: 0.35414796078745714
```

9.3.3 Fit the best estimator on the data

In [12]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, 'lr_ovr_tfidf_hyp_tuned_1_3.pkl')
```

```
Accuracy : 0.0387858347386172
Hamming loss 0.06511174975654942
```

```
Micro-average quality numbers
Precision: 0.3122, Recall: 0.4319, F1-measure: 0.3624
```

```
Macro-average quality numbers
Precision: 0.1628, Recall: 0.2256, F1-measure: 0.1857
```

```
Classification Report
      precision    recall  f1-score   support

0         0.11      0.09      0.10         56
1         0.19      0.40      0.26        129
2         0.17      0.18      0.17         28
3         0.14      0.14      0.14         22
4         0.30      0.56      0.39         18
5         0.09      0.11      0.10         35
```

6	0.24	0.30	0.26	30
7	0.09	0.16	0.12	79
8	0.00	0.00	0.00	8
9	0.10	0.11	0.11	45
10	0.23	0.27	0.25	11
11	0.03	0.03	0.03	40
12	0.10	0.15	0.12	115
13	0.05	0.06	0.05	18
14	0.00	0.00	0.00	9
15	0.08	0.07	0.07	15
16	0.00	0.00	0.00	13
17	0.26	0.41	0.32	368
18	0.14	0.19	0.16	27
19	0.12	0.13	0.12	97
20	0.34	0.50	0.40	551
21	0.05	0.07	0.06	41
22	0.12	0.15	0.13	85
23	0.13	0.10	0.11	42
24	0.10	0.16	0.12	83
25	0.20	0.31	0.24	159
26	0.39	0.45	0.41	112
27	0.00	0.00	0.00	11
28	0.30	0.42	0.35	596
29	0.34	0.57	0.42	190
30	0.31	0.59	0.41	83
31	0.14	0.17	0.15	12
32	0.26	0.42	0.32	26
33	0.15	0.22	0.18	64
34	0.18	0.28	0.22	25
35	0.12	0.10	0.11	29
36	0.25	0.55	0.34	92
37	0.19	0.27	0.22	172
38	0.19	0.26	0.22	136
39	0.21	0.09	0.13	32
40	0.09	0.07	0.08	40
41	0.00	0.00	0.00	5
42	0.09	0.14	0.11	93
43	0.64	0.70	0.67	1155
44	0.11	0.18	0.13	117
45	0.23	0.52	0.32	145
46	0.00	0.00	0.00	5
47	0.24	0.37	0.29	129
48	0.08	0.11	0.10	36
49	0.13	0.14	0.13	44
50	0.08	0.07	0.08	28
51	0.15	0.18	0.16	51
52	0.32	0.43	0.37	395
53	0.13	0.15	0.14	65
54	0.00	0.00	0.00	15
55	0.03	0.03	0.03	37
56	0.32	0.47	0.38	507
57	0.42	0.58	0.49	587
58	0.21	0.26	0.23	148
59	0.21	0.26	0.23	173
60	0.27	0.52	0.35	66
61	0.15	0.18	0.16	51
62	0.04	0.06	0.05	66
63	0.04	0.05	0.04	39
64	0.00	0.00	0.00	8
65	0.22	0.45	0.29	228
66	0.11	0.07	0.09	27
67	0.09	0.11	0.10	119
68	0.55	0.68	0.61	911
69	0.23	0.21	0.22	14
70	0.00	0.00	0.00	13

micro avg	0.31	0.43	0.36	9021
macro avg	0.16	0.23	0.19	9021
weighted avg	0.32	0.43	0.37	9021
samples avg	0.33	0.47	0.33	9021

Time taken to run this cell : 0:00:29.487437

Out[12]:

['lr_ovr_tfidf_hyp_tuned_1_3.pkl']

9.4.1 TFIDF with (1-4 Grams)

In [13]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=50000, smooth_idf=True, norm="l2", tokenizer = lambda x
: x.split(" "), sublinear_tf=False, ngram_range=(1,4))
X_train_multilabel = vectorizer.fit_transform(X_train)
X_test_multilabel = vectorizer.transform(X_test)

#Convert the tags to binary vectors using sklearn's count vectorizer
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)

print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:01:37.292444

9.4.2 Get best estimator using RandomSearch + Logistic Regression

In [14]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
l_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:34:45.812867
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=0.1, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.3400063205214817

9.4.3 Fit the best estimator on the data

In [15]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, 'lr_ovr_tfidf_hyp_tuned_1_4.pkl')
```

Accuracy : 0.023608768971332208
Hamming loss 0.08311996769826378

Micro-average quality numbers
Precision: 0.2569, Recall: 0.4965, F1-measure: 0.3386

Macro-average quality numbers
Precision: 0.1412, Recall: 0.2902, F1-measure: 0.1853

Classification Report

	precision	recall	f1-score	support
0	0.07	0.11	0.09	56
1	0.17	0.51	0.26	129
2	0.11	0.18	0.14	28
3	0.08	0.14	0.10	22
4	0.22	0.61	0.32	18
5	0.05	0.14	0.08	35
6	0.14	0.37	0.20	30
7	0.08	0.28	0.12	79
8	0.00	0.00	0.00	8
9	0.11	0.18	0.13	45
10	0.20	0.27	0.23	11
11	0.04	0.10	0.06	40
12	0.08	0.23	0.12	115
13	0.08	0.17	0.11	18
14	0.00	0.00	0.00	9
15	0.04	0.07	0.05	15
16	0.00	0.00	0.00	13
17	0.23	0.48	0.31	368
18	0.09	0.19	0.12	27
19	0.11	0.26	0.16	97
20	0.32	0.59	0.42	551
21	0.05	0.12	0.08	41
22	0.09	0.31	0.14	85
23	0.07	0.14	0.10	42
24	0.09	0.30	0.14	83
25	0.16	0.39	0.22	159
26	0.34	0.47	0.39	112
27	0.05	0.09	0.06	11
28	0.29	0.49	0.37	596
29	0.27	0.60	0.37	190
30	0.24	0.63	0.34	83
31	0.09	0.17	0.12	12
32	0.20	0.58	0.30	26
33	0.13	0.33	0.19	64
34	0.12	0.28	0.17	25
35	0.08	0.21	0.11	29
36	0.18	0.64	0.28	92
37	0.13	0.31	0.19	172
38	0.14	0.35	0.20	136
39	0.11	0.09	0.10	32

40	0.10	0.15	0.12	40
41	0.00	0.00	0.00	5
42	0.11	0.32	0.16	93
43	0.64	0.72	0.68	1155
44	0.12	0.34	0.18	117
45	0.20	0.65	0.30	145
46	0.20	0.20	0.20	5
47	0.20	0.47	0.28	129
48	0.07	0.14	0.09	36
49	0.06	0.14	0.09	44
50	0.09	0.14	0.11	28
51	0.12	0.31	0.17	51
52	0.32	0.46	0.38	395
53	0.08	0.18	0.11	65
54	0.03	0.07	0.04	15
55	0.06	0.11	0.07	37
56	0.30	0.54	0.39	507
57	0.41	0.63	0.50	587
58	0.15	0.36	0.21	148
59	0.18	0.33	0.23	173
60	0.19	0.62	0.29	66
61	0.09	0.22	0.13	51
62	0.05	0.14	0.07	66
63	0.03	0.08	0.04	39
64	0.00	0.00	0.00	8
65	0.19	0.58	0.29	228
66	0.06	0.07	0.07	27
67	0.07	0.13	0.09	119
68	0.53	0.72	0.61	911
69	0.32	0.43	0.36	14
70	0.00	0.00	0.00	13
micro avg	0.26	0.50	0.34	9021
macro avg	0.14	0.29	0.19	9021
weighted avg	0.30	0.50	0.37	9021
samples avg	0.29	0.53	0.32	9021

Time taken to run this cell : 0:00:17.286450

Out[15]:

['lr_ovr_tfidf_hyp_tuned_1_4.pkl']

10. Taking average number of tags for each movie plots ~ 3

In the EDA section of analysis of tags, we have seen that there are almost 10500 movies which has tags less than or equal to 3.

In [12]:

```
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 3
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

10.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [14]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:04.360477

10.1.2 Get best estimator using RandomSearch + Logistic Regression

In [80]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:08:43.547991

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=16.13975444638871, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5160388863197276

10.1.3 Fit the best estimator on the data

In [82]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_unigram.pkl')
```

Accuracy : 0.3730185497470489
Hamming loss 0.29803260258572234

Micro-average quality numbers
Precision: 0.5021, Recall: 0.5045, F1-measure: 0.5033

Macro-average quality numbers
Precision: 0.4687, Recall: 0.4706, F1-measure: 0.4696

Classification Report				
	precision	recall	f1-score	support
0	0.28	0.28	0.28	596
1	0.59	0.59	0.59	1155
2	0.54	0.54	0.54	911
micro avg	0.50	0.50	0.50	2662
macro avg	0.47	0.47	0.47	2662
weighted avg	0.50	0.50	0.50	2662
samples avg	0.30	0.29	0.28	2662

Time taken to run this cell : 0:00:07.637160

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

'precision', 'predicted', average, warn_for)

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.

'recall', 'true', average, warn_for)

Out[82]:

['3_tags_unigram.pkl']

In []:

10.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [83]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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 : 0:00:23.583663

10.2.2 Get best estimator using RandomSearch + Logistic Regression

In [87]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:01:41.747636
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=210.29664510876157, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.4952942597033721

10.2.3 Fit the best estimator on the data

In [86]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_bigram.pkl')
```

Accuracy : 0.4357504215851602

Hamming loss 0.25362563237774033

Micro-average quality numbers

Precision: 0.5978, Recall: 0.4662, F1-measure: 0.5238

Macro-average quality numbers

Precision: 0.5483, Recall: 0.4233, F1-measure: 0.4713

Classification Report

	precision	recall	f1-score	support
0	0.39	0.18	0.25	596
1	0.65	0.58	0.61	1155
2	0.61	0.50	0.55	911
micro avg	0.60	0.47	0.52	2662
macro avg	0.55	0.42	0.47	2662
weighted avg	0.58	0.47	0.51	2662
samples avg	0.29	0.26	0.26	2662

Time taken to run this cell : 0:00:02.977072

Out[86]:

['3_tags_bigram.pkl']

10.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [88]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:34.491495

10.3.2 Get best estimator using RandomSearch + Logistic Regression

In [89]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:00:40.892219

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=776.6363836807939, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.3947535754694917

10.3.3 Fit the best estimator on the data

In [91]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_trigram.pkl')
```

Accuracy : 0.31770657672849917
Hamming loss 0.33265879707700957

Micro-average quality numbers
Precision: 0.4394, Recall: 0.4046, F1-measure: 0.4213

Macro-average quality numbers
Precision: 0.4013, Recall: 0.3707, F1-measure: 0.3851

Classification Report

	precision	recall	f1-score	support
0	0.23	0.19	0.21	596
1	0.53	0.51	0.52	1155
2	0.45	0.41	0.43	911
micro avg	0.44	0.40	0.42	2662
macro avg	0.40	0.37	0.39	2662
weighted avg	0.43	0.40	0.42	2662
samples avg	0.25	0.24	0.23	2662

Time taken to run this cell : 0:00:01.938548

Out[91]:

['3_tags_trigram.pkl']

In []:

10.4.1 Vectorize the plot synopsis using TFIDF 4grams

In [92]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:35.692424

10.4.2 Get best estimator using RandomSearch + Logistic Regression

In [93]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:00:34.408051
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=987.2532318059303, class_weight=
'balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.28436249765698424
```

10.4.3 Fit the best estimator on the data

In [94]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_quadgram.pkl')
```

Accuracy : 0.36930860033726814
Hamming loss 0.32344013490725126

Micro-average quality numbers
Precision: 0.3947, Recall: 0.1514, F1-measure: 0.2188

Macro-average quality numbers
Precision: 0.3813, Recall: 0.1454, F1-measure: 0.2097

Classification Report

	precision	recall	f1-score	support
0	0.23	0.11	0.15	596
1	0.49	0.17	0.25	1155
2	0.43	0.16	0.23	911
micro avg	0.39	0.15	0.22	2662
macro avg	0.38	0.15	0.21	2662
weighted avg	0.41	0.15	0.22	2662
samples avg	0.08	0.09	0.08	2662

Time taken to run this cell : 0:00:01.620718

Out[94]:

['3_tags_quadgram.pkl']

In []:

10.5.1 Vectorize the plot synopsis using TFIDF Ngrams (1,2)

In [95]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,2))
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 : 0:00:29.006345

10.5.2 Get best estimator using RandomSearch + Logistic Regression

In [96]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:06:16.684811
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=160.98270917946266, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5264291037887373
```

10.5.3 Fit the best estimator on the data

In [97]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_ngrams12.pkl')
```

Accuracy : 0.39494097807757167
Hamming loss 0.2848791455874087

Micro-average quality numbers
Precision: 0.5253, Recall: 0.4996, F1-measure: 0.5121

Macro-average quality numbers
Precision: 0.4847, Recall: 0.4611, F1-measure: 0.4722

Classification Report

	precision	recall	f1-score	support
0	0.29	0.25	0.27	596
1	0.60	0.61	0.60	1155
2	0.57	0.53	0.55	911
micro avg	0.53	0.50	0.51	2662
macro avg	0.48	0.46	0.47	2662
weighted avg	0.52	0.50	0.51	2662
samples avg	0.30	0.29	0.28	2662

Time taken to run this cell : 0:00:06.312505

Out[97]:

['3_tags_ngrams12.pkl']

In []:

10.6.1 Vectorize the plot synopsis using TFIDF Ngrams (1,3)

In [98]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:01:02.880278

10.6.2 Get best estimator using RandomSearch + Logistic Regression

In [99]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:07:36.723844
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=116.8546229723123, class_weight=
'balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5279890472817385
```

10.6.3 Fit the best estimator on the data

In [100]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_ngrams13.pkl')
```

Accuracy : 0.396964586846543

Hamming loss 0.28454187745924675

Micro-average quality numbers

Precision: 0.5258, Recall: 0.5015, F1-measure: 0.5134

Macro-average quality numbers

Precision: 0.4856, Recall: 0.4632, F1-measure: 0.4738

Classification Report

	precision	recall	f1-score	support
0	0.29	0.25	0.27	596
1	0.60	0.61	0.60	1155
2	0.57	0.53	0.55	911
micro avg	0.53	0.50	0.51	2662
macro avg	0.49	0.46	0.47	2662
weighted avg	0.52	0.50	0.51	2662
samples avg	0.30	0.29	0.28	2662

Time taken to run this cell : 0:00:06.153483

Out[100]:

['3_tags_ngrams13.pkl']

In []:

10.7.1 Vectorize the plot synopsis using TFIDF Ngrams (1,4)

In [102]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,4))
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 : 0:01:37.173883

10.7.2 Get best estimator using RandomSearch + Logistic Regression

In [103]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:07:34.542658
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=233.72196849516303, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.525654372835437
```

10.7.3 Fit the best estimator on the data

In [105]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '3_tags_ngrams14.pkl')
```

Accuracy : 0.393929173693086
Hamming loss 0.28566610455311975

Micro-average quality numbers
Precision: 0.5238, Recall: 0.5000, F1-measure: 0.5116

Macro-average quality numbers
Precision: 0.4837, Recall: 0.4617, F1-measure: 0.4721

Classification Report

	precision	recall	f1-score	support
0	0.28	0.25	0.27	596
1	0.60	0.61	0.60	1155
2	0.57	0.53	0.55	911
micro avg	0.52	0.50	0.51	2662
macro avg	0.48	0.46	0.47	2662
weighted avg	0.52	0.50	0.51	2662
samples avg	0.30	0.29	0.28	2662

Time taken to run this cell : 0:00:07.822810

Out[105]:

['3_tags_ngrams14.pkl']

In [106]:

```
##Check
```

11. Taking average number of tags for each movie plots ~ 4

In the EDA section of analysis of tags, we have seen that there are almost 11500 movies which has tags less than or equal to 4.

In [110]:

```
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 4
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=4).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

11.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [111]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:04.547426

11.1.2 Get best estimator using RandomSearch + Logistic Regression

In [112]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:07:55.441729

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1.7402759372625587, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5540249072690279

11.1.3 Fit the best estimator on the data

In [113]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_unigram.pkl')
```

Accuracy : 0.29106239460371

Hamming loss 0.27436762225969646

Micro-average quality numbers

Precision: 0.4994, Recall: 0.6202, F1-measure: 0.5533

Macro-average quality numbers

Precision: 0.4785, Recall: 0.5963, F1-measure: 0.5294

Classification Report

	precision	recall	f1-score	support
0	0.31	0.44	0.36	596
1	0.64	0.69	0.66	1155
2	0.42	0.58	0.49	587
3	0.55	0.68	0.60	911
micro avg	0.50	0.62	0.55	3249
macro avg	0.48	0.60	0.53	3249
weighted avg	0.51	0.62	0.56	3249
samples avg	0.40	0.44	0.39	3249

Time taken to run this cell : 0:00:01.634362

Out[113]:

['4_tags_unigram.pkl']

In []:

11.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [114]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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 : 0:00:22.595376

11.2.2 Get best estimator using RandomSearch + Logistic Regression

In [115]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:02:28.628601
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=12.42955151524594, class_weight=
'balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.48719459337968507
```

11.2.3 Fit the best estimator on the data

In [116]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_bigram.pkl')
```

Accuracy : 0.3399662731871838
Hamming loss 0.24139966273187183

Micro-average quality numbers
Precision: 0.5739, Recall: 0.4611, F1-measure: 0.5114

Macro-average quality numbers
Precision: 0.5298, Recall: 0.4199, F1-measure: 0.4642

Classification Report					
	precision	recall	f1-score	support	
0	0.39	0.21	0.27	596	
1	0.65	0.60	0.62	1155	
2	0.48	0.35	0.41	587	
3	0.60	0.52	0.55	911	
micro avg	0.57	0.46	0.51	3249	
macro avg	0.53	0.42	0.46	3249	
weighted avg	0.56	0.46	0.50	3249	
samples avg	0.35	0.32	0.31	3249	

Time taken to run this cell : 0:00:01.203769

Out[116]:

['4_tags_bigram.pkl']

11.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [117]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:33.718240

11.3.2 Get best estimator using RandomSearch + Logistic Regression

In [118]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:00:56.538398

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=838.7350182525937, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.37183894047483923

11.3.3 Fit the best estimator on the data

In [119]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_trigram.pkl')
```

Accuracy : 0.22327150084317032
Hamming loss 0.318212478920742

Micro-average quality numbers
Precision: 0.4121, Recall: 0.3789, F1-measure: 0.3948

Macro-average quality numbers
Precision: 0.3729, Recall: 0.3435, F1-measure: 0.3574

Classification Report				
	precision	recall	f1-score	support
0	0.23	0.19	0.21	596
1	0.53	0.51	0.52	1155
2	0.29	0.26	0.27	587
3	0.45	0.41	0.43	911
micro avg	0.41	0.38	0.39	3249
macro avg	0.37	0.34	0.36	3249
weighted avg	0.41	0.38	0.39	3249
samples avg	0.28	0.27	0.25	3249

Time taken to run this cell : 0:00:02.146136

Out[119]:

['4_tags_trigram.pkl']

In []:

11.4.1 Vectorize the plot synopsis using TFIDF 4Grams

In [120]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:35.276682

11.4.2 Get best estimator using RandomSearch + Logistic Regression

In [121]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:01:04.681859

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=853.1161249797331, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.26785682789077364

11.4.3 Fit the best estimator on the data

In [122]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_quadgram.pkl')
```

Accuracy : 0.26812816188870153
Hamming loss 0.3022765598650928

Micro-average quality numbers
Precision: 0.3710, Recall: 0.1487, F1-measure: 0.2123

Macro-average quality numbers
Precision: 0.3571, Recall: 0.1431, F1-measure: 0.2034

Classification Report				
	precision	recall	f1-score	support
0	0.23	0.11	0.15	596
1	0.49	0.17	0.25	1155
2	0.28	0.14	0.18	587
3	0.43	0.16	0.23	911
micro avg	0.37	0.15	0.21	3249
macro avg	0.36	0.14	0.20	3249
weighted avg	0.39	0.15	0.21	3249
samples avg	0.09	0.11	0.09	3249

Time taken to run this cell : 0:00:02.184850

Out[122]:

['4_tags_quadgram.pkl']

In []:

11.5.1 Vectorize the plot synopsis using TFIDF Ngrams (1,2)

In [123]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,2))
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 : 0:00:27.781572

11.5.2 Get best estimator using RandomSearch + Logistic Regression

In [124]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:08:55.331291
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=194.24227837639842, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.4996865076447294

11.5.3 Fit the best estimator on the data

In [125]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_ngrams12.pkl')
```

Accuracy : 0.3038785834738617
Hamming loss 0.2702360876897133

Micro-average quality numbers
Precision: 0.5071, Recall: 0.4838, F1-measure: 0.4952

Macro-average quality numbers
Precision: 0.4702, Recall: 0.4488, F1-measure: 0.4589

Classification Report				
	precision	recall	f1-score	support
0	0.29	0.25	0.27	596
1	0.60	0.61	0.60	1155
2	0.43	0.41	0.42	587
3	0.57	0.53	0.55	911
micro avg	0.51	0.48	0.50	3249
macro avg	0.47	0.45	0.46	3249
weighted avg	0.50	0.48	0.49	3249
samples avg	0.35	0.34	0.33	3249

Time taken to run this cell : 0:00:07.921943

Out[125]:

['4_tags_ngrams12.pkl']

In []:

11.6.1 Vectorize the plot synopsis using TFIDF Ngrams (1,3)

In [126]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:01:01.600375

11.6.2 Get best estimator using RandomSearch + Logistic Regression

In [127]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:08:39.527220
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=113.57994837111474, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.5021358090160865

11.6.3 Fit the best estimator on the data

In [128]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_ngrams13.pkl')
```

Accuracy : 0.30522765598650925
Hamming loss 0.26956155143338956

Micro-average quality numbers
Precision: 0.5084, Recall: 0.4857, F1-measure: 0.4968

Macro-average quality numbers
Precision: 0.4716, Recall: 0.4507, F1-measure: 0.4606

Classification Report				
	precision	recall	f1-score	support
0	0.29	0.25	0.27	596
1	0.60	0.61	0.60	1155
2	0.43	0.41	0.42	587
3	0.57	0.53	0.55	911
micro avg	0.51	0.49	0.50	3249
macro avg	0.47	0.45	0.46	3249
weighted avg	0.50	0.49	0.49	3249
samples avg	0.35	0.34	0.33	3249

Time taken to run this cell : 0:00:06.583549

Out[128]:

['4_tags_ngrams13.pkl']

In []:

11.7.1 Vectorize the plot synopsis using TFIDF Ngrams (1,4)

In [129]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,4))
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 : 0:01:44.438828

11.7.2 Get best estimator using RandomSearch + Logistic Regression

In [130]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:10:26.100762
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=65.19474195805786, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.5047835638952324

11.7.3 Fit the best estimator on the data

In [131]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '4_tags_ngrams14.pkl')
```

Accuracy : 0.3045531197301855
Hamming loss 0.2699831365935919

Micro-average quality numbers
Precision: 0.5075, Recall: 0.4888, F1-measure: 0.4980

Macro-average quality numbers
Precision: 0.4712, Recall: 0.4544, F1-measure: 0.4625

Classification Report				
	precision	recall	f1-score	support
0	0.28	0.25	0.26	596
1	0.60	0.61	0.60	1155
2	0.43	0.42	0.43	587
3	0.57	0.54	0.55	911
micro avg	0.51	0.49	0.50	3249
macro avg	0.47	0.45	0.46	3249
weighted avg	0.50	0.49	0.50	3249
samples avg	0.35	0.35	0.33	3249

Time taken to run this cell : 0:00:05.106132

Out[131]:

['4_tags_ngrams14.pkl']

In []:

12. Taking average number of tags for each movie plots ~ 5

In the EDA section of analysis of tags, we have seen that there are almost 11500 movies which has tags less than or equal to 5.

In [132]:

```
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 5
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=5).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

12.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [133]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:04.738666

12.1.2 Get best estimator using RandomSearch + Logistic Regression

In [134]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:10:26.578163
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=189.39503122148082, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.45778340400324086

12.1.3 Fit the best estimator on the data

In [135]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_unigram.pkl')
```

Accuracy : 0.22462057335581787
Hamming loss 0.27622259696458684

Micro-average quality numbers
Precision: 0.4612, Recall: 0.4613, F1-measure: 0.4613

Macro-average quality numbers
Precision: 0.4271, Recall: 0.4281, F1-measure: 0.4274

Classification Report				
	precision	recall	f1-score	support
0	0.34	0.34	0.34	551
1	0.27	0.26	0.27	596
2	0.58	0.59	0.59	1155
3	0.39	0.42	0.40	587
4	0.55	0.53	0.54	911
micro avg	0.46	0.46	0.46	3800
macro avg	0.43	0.43	0.43	3800
weighted avg	0.46	0.46	0.46	3800
samples avg	0.35	0.35	0.32	3800

Time taken to run this cell : 0:00:07.275323

Out[135]:

['5_tags_unigram.pkl']

In []:

12.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [136]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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 : 0:00:22.765204

12.2.2 Get best estimator using RandomSearch + Logistic Regression

In [137]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:03:16.743415
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=255.75996209801355, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.43211158649784287
```

12.2.3 Fit the best estimator on the data

In [138]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_bigram.pkl')
```

Accuracy : 0.27386172006745363
Hamming loss 0.23534569983136594

Micro-average quality numbers
Precision: 0.5593, Recall: 0.3861, F1-measure: 0.4568

Macro-average quality numbers
Precision: 0.4967, Recall: 0.3364, F1-measure: 0.3936

Classification Report

	precision	recall	f1-score	support
0	0.39	0.17	0.24	551
1	0.37	0.16	0.23	596
2	0.64	0.58	0.61	1155
3	0.47	0.29	0.36	587
4	0.60	0.49	0.54	911
micro avg	0.56	0.39	0.46	3800
macro avg	0.50	0.34	0.39	3800
weighted avg	0.53	0.39	0.44	3800
samples avg	0.33	0.28	0.28	3800

Time taken to run this cell : 0:00:04.201555

Out[138]:

['5_tags_bigram.pkl']

12.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [139]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:33.862274

12.3.2 Get best estimator using RandomSearch + Logistic Regression

In [140]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:01:13.029371
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=627.5364233578213, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.3457702974588901
```

12.3.3 Fit the best estimator on the data

In [141]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_trigram.pkl')
```

Accuracy : 0.17774030354131534
Hamming loss 0.30259696458684654

Micro-average quality numbers
Precision: 0.3975, Recall: 0.3500, F1-measure: 0.3722

Macro-average quality numbers
Precision: 0.3553, Recall: 0.3125, F1-measure: 0.3318

Classification Report				
	precision	recall	f1-score	support
0	0.29	0.21	0.25	551
1	0.22	0.18	0.20	596
2	0.53	0.51	0.52	1155
3	0.29	0.26	0.27	587
4	0.45	0.41	0.42	911
micro avg	0.40	0.35	0.37	3800
macro avg	0.36	0.31	0.33	3800
weighted avg	0.39	0.35	0.37	3800
samples avg	0.28	0.27	0.25	3800

Time taken to run this cell : 0:00:02.265874

Out[141]:

['5_tags_trigram.pkl']

In []:

12.4.1 Vectorize the plot synopsis using TFIDF 4Grams

In [142]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:35.370575

12.4.2 Get best estimator using RandomSearch + Logistic Regression

In [143]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:01:33.910396
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=792.2247517419978, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.2553663284637132

12.4.3 Fit the best estimator on the data

In [144]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_quadgram.pkl')
```

Accuracy : 0.22327150084317032
Hamming loss 0.28775716694772346

Micro-average quality numbers
Precision: 0.3501, Recall: 0.1432, F1-measure: 0.2032

Macro-average quality numbers
Precision: 0.3341, Recall: 0.1366, F1-measure: 0.1930

Classification Report				
	precision	recall	f1-score	support
0	0.24	0.11	0.15	551
1	0.23	0.11	0.15	596
2	0.49	0.17	0.25	1155
3	0.28	0.14	0.18	587
4	0.43	0.16	0.23	911
micro avg	0.35	0.14	0.20	3800
macro avg	0.33	0.14	0.19	3800
weighted avg	0.37	0.14	0.20	3800
samples avg	0.09	0.11	0.09	3800

Time taken to run this cell : 0:00:02.412277

Out[144]:

['5_tags_quadgram.pkl']

In []:

12.5.1 Vectorize the plot synopsis using TFIDF Ngrams (1,2)

In [145]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,2))
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 : 0:00:27.924188

12.5.2 Get best estimator using RandomSearch + Logistic Regression

In [146]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:12:37.333694
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=369.0790748802356, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.4690688154941002

12.5.3 Fit the best estimator on the data

In [147]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_ngrams12.pkl')
```

Accuracy : 0.23979763912310287
Hamming loss 0.2638785834738617

Micro-average quality numbers
Precision: 0.4843, Recall: 0.4561, F1-measure: 0.4698

Macro-average quality numbers
Precision: 0.4446, Recall: 0.4182, F1-measure: 0.4306

Classification Report				
	precision	recall	f1-score	support
0	0.34	0.31	0.32	551
1	0.29	0.25	0.27	596
2	0.60	0.61	0.60	1155
3	0.43	0.41	0.42	587
4	0.57	0.52	0.55	911
micro avg	0.48	0.46	0.47	3800
macro avg	0.44	0.42	0.43	3800
weighted avg	0.48	0.46	0.47	3800
samples avg	0.35	0.35	0.32	3800

Time taken to run this cell : 0:00:12.003120

Out[147]:

['5_tags_ngrams12.pkl']

In []:

12.6.1 Vectorize the plot synopsis using TFIDF Ngrams (1,3)

In [148]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, 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)
```

Time taken to run this cell : 0:01:02.217223

12.6.2 Get best estimator using RandomSearch + Logistic Regression

In [149]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:13:12.852032
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=308.68277208178677, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.4699172072908497

12.6.3 Fit the best estimator on the data

In [150]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_ngrams13.pkl')
```

Accuracy : 0.23946037099494097
Hamming loss 0.26360876897133223

Micro-average quality numbers
Precision: 0.4849, Recall: 0.4576, F1-measure: 0.4709

Macro-average quality numbers
Precision: 0.4451, Recall: 0.4199, F1-measure: 0.4319

Classification Report				
	precision	recall	f1-score	support
0	0.34	0.30	0.32	551
1	0.29	0.25	0.27	596
2	0.60	0.60	0.60	1155
3	0.43	0.41	0.42	587
4	0.57	0.53	0.55	911
micro avg	0.48	0.46	0.47	3800
macro avg	0.45	0.42	0.43	3800
weighted avg	0.48	0.46	0.47	3800
samples avg	0.35	0.35	0.33	3800

Time taken to run this cell : 0:00:11.010485

Out[150]:

['5_tags_ngrams13.pkl']

In []:

12.7.1 Vectorize the plot synopsis using TFIDF Ngrams (1,4)

In [151]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(min_df=0.00009, max_features=100000, smooth_idf=True, norm="l2", tokenizer = lambda
x: x.split(" "), sublinear_tf=False, ngram_range=(1,4))
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 : 0:01:38.255691

12.7.2 Get best estimator using RandomSearch + Logistic Regression

In [152]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

#alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
alpha=stats.uniform(0,1000)
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:09:33.067521
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=50.11676327193071, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.47899463648002655

12.7.3 Fit the best estimator on the data

In [153]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

from sklearn.externals import joblib
joblib.dump(classifier, '5_tags_ngrams14.pkl')
```

Accuracy : 0.2418212478920742
Hamming loss 0.26408094435075885

Micro-average quality numbers
Precision: 0.4843, Recall: 0.4671, F1-measure: 0.4756

Macro-average quality numbers
Precision: 0.4444, Recall: 0.4294, F1-measure: 0.4366

Classification Report				
	precision	recall	f1-score	support
0	0.34	0.32	0.33	551
1	0.28	0.25	0.26	596
2	0.61	0.61	0.61	1155
3	0.43	0.43	0.43	587
4	0.57	0.54	0.56	911
micro avg	0.48	0.47	0.48	3800
macro avg	0.44	0.43	0.44	3800
weighted avg	0.48	0.47	0.47	3800
samples avg	0.36	0.35	0.33	3800

Time taken to run this cell : 0:00:06.295944

Out[153]:

['5_tags_ngrams14.pkl']

Analysing using Character sequences

13. Taking average number of tags for each movie plots ~ 3

In the EDA section of analysis of tags, we have seen that there are almost 10500 movies which has tags less than or equal to 3.

In [19]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default '
split()' will tokenize each tag using space.
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

#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number
= 3
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

13.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [16]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng
ram_range=(1,1))
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 : 0:00:09.374659

13.1.2 Get best estimator using RandomSearch + Logistic Regression

In [35]:

```
st=datetime.now()

alpha=stats.uniform(0,1000)

penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:16:01.245669
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=463.34877556359345, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)
Best Cross Validation Score: 0.44164340665455953

13.1.3 Fit the best estimator on the data

In [36]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_unigram.pkl')
```

Accuracy : 0.1946037099494098
Hamming loss 0.44890387858347386

Micro-average quality numbers
Precision: 0.3493, Recall: 0.5793, F1-measure: 0.4358

Macro-average quality numbers
Precision: 0.3484, Recall: 0.5760, F1-measure: 0.4278

Classification Report

	precision	recall	f1-score	support
0	0.23	0.55	0.32	596
1	0.45	0.58	0.51	1155
2	0.37	0.60	0.46	911
micro avg	0.35	0.58	0.44	2662
macro avg	0.35	0.58	0.43	2662
weighted avg	0.37	0.58	0.45	2662
samples avg	0.27	0.34	0.28	2662

Time taken to run this cell : 0:00:00.378996

Out[36]:

['3_tags_char_unigram.pkl']

In []:

13.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [37]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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 : 0:00:22.633786

13.2.2 Get best estimator using RandomSearch + Logistic Regression

In [38]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:07:28.341077
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=10, class_weight='balanced', dual=False,
        fit_intercept=True, intercept_scaling=1, max_iter=100,
        multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
        solver='warn', tol=0.0001, verbose=0, warm_start=False),
        n_jobs=-1)
Best Cross Validation Score:  0.5150786664408002
```

13.2.3 Fit the best estimator on the data

In [39]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_bigram.pkl')
```

Accuracy : 0.29376053962900506

Hamming loss 0.35480607082630694

Micro-average quality numbers

Precision: 0.4368, Recall: 0.6416, F1-measure: 0.5198

Macro-average quality numbers

Precision: 0.4342, Recall: 0.6279, F1-measure: 0.5067

Classification Report

	precision	recall	f1-score	support
0	0.26	0.55	0.35	596
1	0.57	0.68	0.62	1155
2	0.47	0.66	0.55	911
micro avg	0.44	0.64	0.52	2662
macro avg	0.43	0.63	0.51	2662
weighted avg	0.47	0.64	0.54	2662
samples avg	0.31	0.37	0.32	2662

Time taken to run this cell : 0:01:08.672116

Out[39]:

['3_tags_char_bigram.pkl']

13.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [40]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:26.574681

13.3.2 Get best estimator using RandomSearch + Logistic Regression

In [41]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:47:15.029775

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=0.1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5521907144586302

13.3.3 Fit the best estimator on the data

In [42]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_trigram.pkl')
```

Accuracy : 0.33929173693086

Hamming loss 0.31849353569421024

Micro-average quality numbers

Precision: 0.4761, Recall: 0.6386, F1-measure: 0.5455

Macro-average quality numbers

Precision: 0.4710, Recall: 0.6238, F1-measure: 0.5302

Classification Report

	precision	recall	f1-score	support
0	0.28	0.53	0.37	596
1	0.63	0.66	0.64	1155
2	0.51	0.68	0.58	911
micro avg	0.48	0.64	0.55	2662
macro avg	0.47	0.62	0.53	2662
weighted avg	0.51	0.64	0.56	2662
samples avg	0.31	0.37	0.32	2662

Time taken to run this cell : 0:00:04.036597

Out[42]:

['3_tags_char_trigram.pkl']

In []:

13.4.1 Vectorize the plot synopsis using TFIDF 4Grams

In [43]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:31.899656

13.4.2 Get best estimator using RandomSearch + Logistic Regression

In [44]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:05:38.154677

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=0.1, class_weight='balanced', dual=False,

```
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.5646043371417068

13.4.3 Fit the best estimator on the data

In [45]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_quadgram.pkl')
```

Accuracy : 0.3625632377740304

Hamming loss 0.30376616076447444

Micro-average quality numbers

Precision: 0.4942, Recall: 0.6409, F1-measure: 0.5581

Macro-average quality numbers

Precision: 0.4824, Recall: 0.6205, F1-measure: 0.5384

Classification Report

	precision	recall	f1-score	support
0	0.29	0.49	0.36	596
1	0.64	0.68	0.66	1155
2	0.52	0.69	0.59	911
micro avg	0.49	0.64	0.56	2662
macro avg	0.48	0.62	0.54	2662
weighted avg	0.52	0.64	0.57	2662
samples avg	0.32	0.37	0.32	2662

Time taken to run this cell : 0:00:12.006572

Out[45]:

['3_tags_char_quadgram.pkl']

In []:

13.5.1 Vectorize the plot synopsis using TFIDF Char 5Grams

In [88]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(5,5))
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 : 0:00:41.294351

13.5.2 Get best estimator using RandomSearch + Logistic Regression

In [89]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:14:53.505203

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5794434846935947

13.5.3 Fit the best estimator on the data

In [90]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams55.pkl')
```

Accuracy : 0.38954468802698144

Hamming loss 0.28566610455311975

Micro-average quality numbers

Precision: 0.5186, Recall: 0.6341, F1-measure: 0.5706

Macro-average quality numbers

Precision: 0.4988, Recall: 0.6088, F1-measure: 0.5466

Classification Report

	precision	recall	f1-score	support
0	0.31	0.46	0.37	596
1	0.64	0.69	0.66	1155
2	0.55	0.67	0.60	911
micro avg	0.52	0.63	0.57	2662
macro avg	0.50	0.61	0.55	2662
weighted avg	0.53	0.63	0.58	2662
samples avg	0.32	0.37	0.33	2662

Time taken to run this cell : 0:00:48.453698

Out[90]:

['3_tags_char_5grams.pkl']

In []:

13.6.1 Vectorize the plot synopsis using TFIDF Char 6 grams

In [18]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(6,6))
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 : 0:00:46.200840

13.6.2 Get best estimator using RandomSearch + Logistic Regression

In [92]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:05:53.640614

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.577594690298686

13.6.3 Fit the best estimator on the data

In [93]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams66.pkl')
```

Accuracy : 0.3858347386172007

Hamming loss 0.28757729061270376

Micro-average quality numbers

Precision: 0.5153, Recall: 0.6570, F1-measure: 0.5776

Macro-average quality numbers

Precision: 0.5030, Recall: 0.6372, F1-measure: 0.5580

Classification Report

	precision	recall	f1-score	support
0	0.31	0.52	0.39	596
1	0.66	0.70	0.68	1155
2	0.54	0.68	0.60	911
micro avg	0.52	0.66	0.58	2662
macro avg	0.50	0.64	0.56	2662
weighted avg	0.54	0.66	0.59	2662
samples avg	0.33	0.39	0.33	2662

Time taken to run this cell : 0:00:09.229328

Out[93]:

['3_tags_char_ngrams13.pkl']

13.7.1 Vectorize the plot synopsis using TFIDF 3-6Grams

In [20]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(3,6))
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 : 0:01:29.306033

13.7.2 Get best estimator using RandomSearch + Logistic Regression

In [21]:

```
import warnings
warnings.filterwarnings("ignore")
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:13:19.140348

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

```
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.5814690761358954

13.7.3 Fit the best estimator on the data

In [22]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams36.pkl')
```

Accuracy : 0.3824620573355818
Hamming loss 0.2896008993816751

Micro-average quality numbers
Precision: 0.5128, Recall: 0.6491, F1-measure: 0.5729

Macro-average quality numbers
Precision: 0.4970, Recall: 0.6273, F1-measure: 0.5517

Classification Report

	precision	recall	f1-score	support
0	0.31	0.50	0.38	596
1	0.64	0.70	0.67	1155
2	0.54	0.68	0.60	911
avg / total	0.53	0.65	0.58	2662

Time taken to run this cell : 0:00:14.901985

Out[22]:

```
['3_tags_char_ngrams36.pkl']
```

13.8.1 Vectorize the plot synopsis using TFIDF 1-6Grams

In [28]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(1,6))
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 : 0:01:45.353974

13.8.2 Get best estimator using RandomSearch + Logistic Regression

In [29]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 0:27:44.117895
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=10, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.5583892906664927
```

13.8.3 Fit the best estimator on the data

In [30]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams16.pkl')
```

Accuracy : 0.3672849915682968

Hamming loss 0.29926925238898255

Micro-average quality numbers

Precision: 0.5000, Recall: 0.6011, F1-measure: 0.5459

Macro-average quality numbers

Precision: 0.4784, Recall: 0.5733, F1-measure: 0.5203

Classification Report

	precision	recall	f1-score	support
0	0.29	0.41	0.34	596
1	0.61	0.66	0.64	1155
2	0.53	0.65	0.59	911
avg / total	0.51	0.60	0.55	2662

Time taken to run this cell : 0:02:53.205525

Out[30]:

['3_tags_char_ngrams16.pkl']

In []:

13.9.1 Vectorize the plot synopsis using TFIDF 3-4Grams

In [23]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(3,4))
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 : 0:00:28.393619

13.9.2 Get best estimator using RandomSearch + Logistic Regression

In [24]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 0:10:23.147088
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.5737365989364919
```

13.9.3 Fit the best estimator on the data

In [25]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams34.pkl')
```

Accuracy : 0.3790893760539629

Hamming loss 0.29094997189432265

Micro-average quality numbers

Precision: 0.5113, Recall: 0.6273, F1-measure: 0.5634

Macro-average quality numbers

Precision: 0.4946, Recall: 0.6038, F1-measure: 0.5409

Classification Report

	precision	recall	f1-score	support
0	0.30	0.46	0.36	596
1	0.64	0.68	0.66	1155
2	0.54	0.67	0.60	911
avg / total	0.53	0.63	0.57	2662

Time taken to run this cell : 0:00:10.436067

Out[25]:

['3_tags_char_ngrams34.pkl']

14. Taking average number of tags for each movie plots ~ 4

In the EDA section of analysis of tags, we have seen that there are almost 11500 movies which has tags less than or equal to 4.

In [26]:

```
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 4
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=4).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

14.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [72]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:08.959338

14.1.2 Get best estimator using RandomSearch + Logistic Regression

In [73]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:08:16.251276

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1e-06, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.4397292479326727

14.1.3 Fit the best estimator on the data

In [74]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_unigram.pkl')
```

Accuracy : 0.048903878583473864
Hamming loss 0.5850758853288365

Micro-average quality numbers
Precision: 0.3011, Recall: 0.8593, F1-measure: 0.4459

Macro-average quality numbers
Precision: 0.3071, Recall: 0.8129, F1-measure: 0.4173

Classification Report

	precision	recall	f1-score	support
0	0.20	1.00	0.33	596
1	0.39	0.97	0.56	1155
2	0.33	0.28	0.30	587
3	0.31	1.00	0.47	911
micro avg	0.30	0.86	0.45	3249
macro avg	0.31	0.81	0.42	3249
weighted avg	0.32	0.86	0.45	3249
samples avg	0.30	0.59	0.38	3249

Time taken to run this cell : 0:00:00.876526

Out[74]:

['4_tags_char_unigram.pkl']

In []:

14.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [75]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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 : 0:00:24.301227

14.2.2 Get best estimator using RandomSearch + Logistic Regression

In [76]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 0:05:36.406115
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.4980526671896285
```

14.2.3 Fit the best estimator on the data

In [77]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_bigram.pkl')
```

Accuracy : 0.1966273187183811
Hamming loss 0.3473018549747049

Micro-average quality numbers
Precision: 0.4135, Recall: 0.6402, F1-measure: 0.5025

Macro-average quality numbers
Precision: 0.4101, Recall: 0.6300, F1-measure: 0.4902

Classification Report

	precision	recall	f1-score	support
0	0.25	0.54	0.34	596
1	0.57	0.67	0.62	1155
2	0.35	0.64	0.45	587
3	0.47	0.67	0.55	911
micro avg	0.41	0.64	0.50	3249
macro avg	0.41	0.63	0.49	3249
weighted avg	0.44	0.64	0.52	3249
samples avg	0.36	0.46	0.38	3249

Time taken to run this cell : 0:00:02.079338

Out[77]:

['4_tags_char_bigram.pkl']

14.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [78]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:25.615812

14.3.2 Get best estimator using RandomSearch + Logistic Regression

In [79]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:34:12.521707
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5438865706060322
```

14.3.3 Fit the best estimator on the data

In [80]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_trigram.pkl')
```

Accuracy : 0.2650927487352445

Hamming loss 0.29527824620573356

Micro-average quality numbers

Precision: 0.4713, Recall: 0.6393, F1-measure: 0.5426

Macro-average quality numbers

Precision: 0.4611, Recall: 0.6255, F1-measure: 0.5258

Classification Report

	precision	recall	f1-score	support
0	0.29	0.52	0.38	596
1	0.63	0.67	0.65	1155
2	0.39	0.62	0.48	587
3	0.53	0.70	0.60	911
micro avg	0.47	0.64	0.54	3249
macro avg	0.46	0.63	0.53	3249
weighted avg	0.50	0.64	0.55	3249
samples avg	0.39	0.46	0.39	3249

Time taken to run this cell : 0:00:11.880445

Out[80]:

['4_tags_char_trigram.pkl']

In []:

14.4.1 Vectorize the plot synopsis using TFIDF 4Grams

In [81]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:41.507473

14.4.2 Get best estimator using RandomSearch + Logistic Regression

In [82]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:17:22.292227

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.554634460901214

14.4.3 Fit the best estimator on the data

In [83]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_quadgram.pkl')
```

Accuracy : 0.2775716694772344
Hamming loss 0.2814502529510961

Micro-average quality numbers
Precision: 0.4896, Recall: 0.6464, F1-measure: 0.5572

Macro-average quality numbers
Precision: 0.4761, Recall: 0.6295, F1-measure: 0.5379

Classification Report

	precision	recall	f1-score	support
0	0.31	0.51	0.38	596
1	0.65	0.69	0.67	1155
2	0.40	0.62	0.49	587
3	0.54	0.70	0.61	911
micro avg	0.49	0.65	0.56	3249
macro avg	0.48	0.63	0.54	3249
weighted avg	0.51	0.65	0.57	3249
samples avg	0.40	0.46	0.40	3249

Time taken to run this cell : 0:00:12.910091

Out[83]:

['4_tags_char_quadgram.pkl']

In []:

14.5.1 Vectorize the plot synopsis using TFIDF 5Grams

In [95]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(5,5))
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 : 0:00:42.594618

14.5.2 Get best estimator using RandomSearch + Logistic Regression

In [96]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:29:27.363661

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.558657925648389

14.5.3 Fit the best estimator on the data

In [97]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_ngrams55.pkl')
```

Accuracy : 0.2957841483979764
Hamming loss 0.2732715008431703

Micro-average quality numbers
Precision: 0.5010, Recall: 0.6242, F1-measure: 0.5558

Macro-average quality numbers
Precision: 0.4813, Recall: 0.6014, F1-measure: 0.5331

Classification Report

	precision	recall	f1-score	support
0	0.31	0.46	0.37	596
1	0.64	0.69	0.66	1155
2	0.43	0.58	0.49	587
3	0.55	0.67	0.60	911
micro avg	0.50	0.62	0.56	3249
macro avg	0.48	0.60	0.53	3249
weighted avg	0.51	0.62	0.56	3249
samples avg	0.39	0.44	0.39	3249

Time taken to run this cell : 0:00:52.283178

Out[97]:

['4_tags_char_ngrams55.pkl']

In []:

14.6.1 Vectorize the plot synopsis using TFIDF 6Grams

In [98]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(6,6))
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 : 0:01:48.372274

14.6.2 Get best estimator using RandomSearch + Logistic Regression

In [99]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:09:15.906938

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l1', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5575921569529182

14.6.3 Fit the best estimator on the data

In [100]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_ngrams66.pkl')
```

Accuracy : 0.2836424957841484

Hamming loss 0.27622259696458684

Micro-average quality numbers

Precision: 0.4968, Recall: 0.6519, F1-measure: 0.5639

Macro-average quality numbers

Precision: 0.4834, Recall: 0.6350, F1-measure: 0.5452

Classification Report

	precision	recall	f1-score	support
0	0.31	0.52	0.39	596
1	0.66	0.70	0.68	1155
2	0.42	0.63	0.51	587
3	0.54	0.68	0.60	911
micro avg	0.50	0.65	0.56	3249
macro avg	0.48	0.64	0.55	3249
weighted avg	0.52	0.65	0.57	3249
samples avg	0.40	0.47	0.40	3249

Time taken to run this cell : 0:00:08.045899

Out[100]:

['4_tags_char_ngrams66.pkl']

14.7.1 Vectorize the plot synopsis using TFIDF 3-6Grams

In [27]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(3,6))
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 : 0:01:23.887635

14.7.2 Get best estimator using RandomSearch + Logistic Regression

In [28]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:32:24.630346
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=0.01, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5451120850790291
```

14.7.3 Fit the best estimator on the data

In [29]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_ngrams36.pkl')
```

Accuracy : 0.23170320404721753

Hamming loss 0.31011804384485664

Micro-average quality numbers

Precision: 0.4552, Recall: 0.6713, F1-measure: 0.5425

Macro-average quality numbers

Precision: 0.4499, Recall: 0.6593, F1-measure: 0.5283

Classification Report

	precision	recall	f1-score	support
0	0.28	0.58	0.37	596
1	0.63	0.70	0.66	1155
2	0.39	0.65	0.49	587
3	0.50	0.71	0.59	911
avg / total	0.49	0.67	0.56	3249

Time taken to run this cell : 0:00:09.234611

Out[29]:

['4_tags_char_ngrams36.pkl']

14.8.1 Vectorize the plot synopsis using TFIDF 1-6Grams

In [32]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(1,6))
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 : 0:01:39.524167

14.8.2 Get best estimator using RandomSearch + Logistic Regression

In [33]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:33:22.151960
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5565907439978898
```

14.8.3 Fit the best estimator on the data

In [34]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_ngrams16.pkl')
```

Accuracy : 0.2650927487352445

Hamming loss 0.29064080944350756

Micro-average quality numbers

Precision: 0.4778, Recall: 0.6559, F1-measure: 0.5529

Macro-average quality numbers

Precision: 0.4671, Recall: 0.6404, F1-measure: 0.5357

Classification Report

	precision	recall	f1-score	support
0	0.29	0.54	0.38	596
1	0.64	0.70	0.67	1155
2	0.41	0.63	0.50	587
3	0.52	0.69	0.60	911
avg / total	0.50	0.66	0.56	3249

Time taken to run this cell : 0:00:17.636344

Out[34]:

['4_tags_char_ngrams16.pkl']

14.9.1 Vectorize the plot synopsis using TFIDF 3-4Grams

In [30]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng
ram_range=(3,4))
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 : 0:00:28.285770

14.9.2 Get best estimator using RandomSearch + Logistic Regression

In [31]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:22:26.363969

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

```
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.5520061427303079

14.9.3 Fit the best estimator on the data

In [32]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '4_tags_char_ngrams34.pkl')
```

Accuracy : 0.2863406408094435

Hamming loss 0.27951096121416524

Micro-average quality numbers

Precision: 0.4919, Recall: 0.6205, F1-measure: 0.5488

Macro-average quality numbers

Precision: 0.4749, Recall: 0.6002, F1-measure: 0.5276

Classification Report

	precision	recall	f1-score	support
0	0.30	0.46	0.36	596
1	0.64	0.68	0.66	1155
2	0.42	0.59	0.49	587
3	0.54	0.67	0.60	911
avg / total	0.51	0.62	0.56	3249

Time taken to run this cell : 0:00:13.558885

Out[32]:

['4_tags_char_ngrams34.pkl']

15. Taking average number of tags for each movie plots ~ 5

In the EDA section of analysis of tags, we have seen that there are almost 11500 movies which has tags less than or equal to 5.

In [33]:

```
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 5
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=5).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

15.1.1 Vectorize the plot synopsis using TFIDF Unigrams

In [34]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(1,1))
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 : 0:00:04.122752

15.1.2 Get best estimator using RandomSearch + Logistic Regression

In [35]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:03:26.402614

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=10000, class_weight='balanced', dual=False, fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.40945364775743637

15.1.3 Fit the best estimator on the data

In [36]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_unigram.pkl')
```

Accuracy : 0.07487352445193929
Hamming loss 0.4388532883642496

Micro-average quality numbers
Precision: 0.3107, Recall: 0.5845, F1-measure: 0.4057

Macro-average quality numbers
Precision: 0.3085, Recall: 0.5842, F1-measure: 0.3971

Classification Report

	precision	recall	f1-score	support
0	0.24	0.61	0.34	551
1	0.22	0.55	0.32	596
2	0.45	0.58	0.51	1155
3	0.27	0.58	0.36	587
4	0.37	0.60	0.46	911
avg / total	0.33	0.58	0.42	3800

Time taken to run this cell : 0:00:00.577143

Out[36]:

['5_tags_char_unigram.pkl']

In []:

15.2.1 Vectorize the plot synopsis using TFIDF Bigrams

In [37]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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 : 0:00:09.429233

15.2.2 Get best estimator using RandomSearch + Logistic Regression

In [38]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 1:01:06.701920
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=10, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.46984567499676855
```

15.2.3 Fit the best estimator on the data

In [39]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_bigram.pkl')
```

Accuracy : 0.13288364249578416

Hamming loss 0.35291736930860035

Micro-average quality numbers

Precision: 0.3857, Recall: 0.6355, F1-measure: 0.4800

Macro-average quality numbers

Precision: 0.3823, Recall: 0.6257, F1-measure: 0.4669

Classification Report

	precision	recall	f1-score	support
--	-----------	--------	----------	---------

0	0.27	0.63	0.38	551
1	0.25	0.54	0.34	596
2	0.58	0.67	0.62	1155
3	0.34	0.63	0.44	587
4	0.47	0.66	0.55	911

avg / total	0.42	0.64	0.50	3800
-------------	------	------	------	------

Time taken to run this cell : 0:00:03.213662

Out[39]:

['5_tags_char_bigram.pkl']

15.3.1 Vectorize the plot synopsis using TFIDF Trigrams

In [40]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, 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)
```

Time taken to run this cell : 0:00:11.855936

15.3.2 Get best estimator using RandomSearch + Logistic Regression

In [41]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 1:00:15.565148
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
fit_intercept=True, intercept_scaling=1, max_iter=100,
multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
n_jobs=-1)
Best Cross Validation Score: 0.5131329825480064
```

15.3.3 Fit the best estimator on the data

In [42]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_trigram.pkl')
```

Accuracy : 0.19865092748735244
Hamming loss 0.2959865092748735

Micro-average quality numbers
Precision: 0.4446, Recall: 0.6211, F1-measure: 0.5182

Macro-average quality numbers
Precision: 0.4325, Recall: 0.6038, F1-measure: 0.4995

Classification Report

	precision	recall	f1-score	support
0	0.31	0.56	0.39	551
1	0.29	0.50	0.37	596
2	0.62	0.67	0.65	1155
3	0.41	0.60	0.49	587
4	0.53	0.69	0.60	911
avg / total	0.47	0.62	0.53	3800

Time taken to run this cell : 0:00:07.073168

Out[42]:

['5_tags_char_trigram.pkl']

In []:

15.4.1 Vectorize the plot synopsis using TFIDF 4Grams

In [43]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(4,4))
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 : 0:00:14.785512

15.4.2 Get best estimator using RandomSearch + Logistic Regression

In [44]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:22:13.901253

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1, penalty='l2', random_state=None, solver='liblinear', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5249801838932145

15.4.3 Fit the best estimator on the data

In [45]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_quadgram.pkl')
```

Accuracy : 0.22360876897133222
Hamming loss 0.27521079258010117

Micro-average quality numbers
Precision: 0.4711, Recall: 0.6003, F1-measure: 0.5279

Macro-average quality numbers
Precision: 0.4518, Recall: 0.5775, F1-measure: 0.5045

Classification Report

	precision	recall	f1-score	support
0	0.34	0.51	0.41	551
1	0.30	0.44	0.36	596
2	0.65	0.68	0.66	1155
3	0.42	0.59	0.49	587
4	0.54	0.67	0.60	911
avg / total	0.49	0.60	0.54	3800

Time taken to run this cell : 0:00:11.819706

Out[45]:

['5_tags_char_quadgram.pkl']

In []:

In []:

15.5.1 Vectorize the plot synopsis using TFIDF 5Grams

In [102]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(5,5))
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 : 0:00:53.274324

15.5.2 Get best estimator using RandomSearch + Logistic Regression

In [103]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:32:46.797805

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

```
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
    solver='warn', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.522676124452314

15.5.3 Fit the best estimator on the data

In [104]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_ngrams55.pkl')
```

Accuracy : 0.1993254637436762
Hamming loss 0.2855986509274874

Micro-average quality numbers
Precision: 0.4590, Recall: 0.6392, F1-measure: 0.5343

Macro-average quality numbers
Precision: 0.4464, Recall: 0.6207, F1-measure: 0.5143

Classification Report

	precision	recall	f1-score	support
0	0.30	0.58	0.40	551
1	0.32	0.52	0.40	596
2	0.66	0.70	0.68	1155
3	0.41	0.61	0.49	587
4	0.54	0.69	0.60	911
micro avg	0.46	0.64	0.53	3800
macro avg	0.45	0.62	0.51	3800
weighted avg	0.49	0.64	0.55	3800
samples avg	0.39	0.49	0.40	3800

Time taken to run this cell : 0:00:38.503965

Out[104]:

['5_tags_char_ngrams55.pkl']

In []:

15.6.1 Vectorize the plot synopsis using TFIDF 6Grams

In [105]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(6,6))
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 : 0:02:03.733189

15.6.2 Get best estimator using RandomSearch + Logistic Regression

In [106]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:16:41.081102

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

fit_intercept=True, intercept_scaling=1, max_iter=100, multi_class='warn', n_jobs=None, penalty='l2', random_state=None, solver='warn', tol=0.0001, verbose=0, warm_start=False), n_jobs=-1)

Best Cross Validation Score: 0.5308788296165124

15.6.3 Fit the best estimator on the data

In [107]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_ngrams66.pkl')
```

Accuracy : 0.224283305227656

Hamming loss 0.27237774030354134

Micro-average quality numbers

Precision: 0.4754, Recall: 0.6047, F1-measure: 0.5323

Macro-average quality numbers

Precision: 0.4540, Recall: 0.5790, F1-measure: 0.5071

Classification Report

	precision	recall	f1-score	support
0	0.33	0.49	0.40	551
1	0.31	0.46	0.37	596
2	0.64	0.69	0.67	1155
3	0.43	0.58	0.49	587
4	0.55	0.68	0.61	911
micro avg	0.48	0.60	0.53	3800
macro avg	0.45	0.58	0.51	3800
weighted avg	0.49	0.60	0.54	3800
samples avg	0.39	0.46	0.39	3800

Time taken to run this cell : 0:00:23.216361

Out[107]:

['4_tags_char_ngrams66.pkl']

In []:

In []:

15.7.1 Vectorize the plot synopsis using TFIDF 3-6Grams

In [46]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(3,6))
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 : 0:01:24.567291

15.7.2 Get best estimator using RandomSearch + Logistic Regression

In [47]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 0:20:59.710166

Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,

```
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l2', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.5325486379443722

15.7.3 Fit the best estimator on the data

In [48]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_ngrams36.pkl')
```

Accuracy : 0.21854974704890387
Hamming loss 0.2793929173693086

Micro-average quality numbers
Precision: 0.4664, Recall: 0.6250, F1-measure: 0.5342

Macro-average quality numbers
Precision: 0.4490, Recall: 0.6033, F1-measure: 0.5121

Classification Report

	precision	recall	f1-score	support
0	0.33	0.53	0.41	551
1	0.31	0.50	0.38	596
2	0.64	0.70	0.67	1155
3	0.42	0.60	0.50	587
4	0.54	0.68	0.60	911
avg / total	0.49	0.62	0.54	3800

Time taken to run this cell : 0:00:22.767076

Out[48]:

['5_tags_char_ngrams36.pkl']

In []:

15.8.1 Vectorize the plot synopsis using TFIDF 1-6Grams

In [36]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng
ram_range=(1,6))
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 : 0:01:40.941822

15.8.2 Get best estimator using RandomSearch + Logistic Regression

In [37]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]
penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 1:24:43.077360
Best estimator: OneVsRestClassifier(estimator=LogisticRegression(C=10, class_weight='balanced', dual=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score: 0.5035367869358058
```

15.8.3 Fit the best estimator on the data

In [38]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_ngrams16.pkl')
```

Accuracy : 0.20269814502529512

Hamming loss 0.28836424957841483

Micro-average quality numbers

Precision: 0.4506, Recall: 0.5695, F1-measure: 0.5031

Macro-average quality numbers

Precision: 0.4281, Recall: 0.5419, F1-measure: 0.4768

Classification Report

	precision	recall	f1-score	support
0	0.32	0.47	0.38	551
1	0.29	0.41	0.34	596
2	0.61	0.66	0.64	1155
3	0.39	0.52	0.44	587
4	0.53	0.65	0.59	911
avg / total	0.47	0.57	0.51	3800

Time taken to run this cell : 0:03:49.840338

Out[38]:

['5_tags_char_ngrams16.pkl']

In []:

15.9.1 Vectorize the plot synopsis using TFIDF 3-4Grams

In [49]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng
ram_range=(3,4))
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 : 0:00:27.966497

15.9.2 Get best estimator using RandomSearch + Logistic Regression

In [50]:

```
import warnings
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:25:42.279223
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual
=False,
    fit_intercept=True, intercept_scaling=1, max_iter=100,
    multi_class='ovr', n_jobs=1, penalty='l1', random_state=None,
    solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
    n_jobs=-1)
Best Cross Validation Score:  0.5207662759679801
```

15.9.3 Fit the best estimator on the data

In [51]:

```
import warnings
warnings.filterwarnings("ignore")

start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '5_tags_char_ngrams34.pkl')
```

Accuracy : 0.19190556492411467
Hamming loss 0.29133220910623947

Micro-average quality numbers
Precision: 0.4514, Recall: 0.6350, F1-measure: 0.5277

Macro-average quality numbers
Precision: 0.4423, Recall: 0.6204, F1-measure: 0.5103

Classification Report

	precision	recall	f1-score	support
0	0.30	0.58	0.40	551
1	0.31	0.52	0.39	596
2	0.65	0.68	0.67	1155
3	0.41	0.63	0.50	587
4	0.54	0.69	0.60	911
avg / total	0.48	0.64	0.54	3800

Time taken to run this cell : 0:00:07.434768

Out[51]:

['5_tags_char_ngrams34.pkl']

In []:

16. ML Models with Binary Relevance + Taking average number of tags for each movie plots ~ 3

In the EDA section of analysis of tags, we have seen that there are almost 10500 movies which has tags less than or equal to 3.

In [5]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'sp
lit()' will tokenize each tag using space.
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

#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number =
3
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

16.1.1 Vectorize the plot synopsis using TFIDF Char 6 grams

In [7]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng
ram_range=(6,6))
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 : 0:00:48.206001

Hyper-parameter tuning section

In [6]:

```
#Refer: http://scikit.ml/api/skmultilearn.problem\_transform.br.html
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC

start = datetime.now()

parameters = [
    {
        'classifier': [MultinomialNB()],
        'classifier__alpha': [0.5,0.7,0.9]
    },
    {
        'classifier': [GaussianNB()],
        'classifier__alpha': [0.5,0.7,0.9],
    },
    {
        'classifier': [LogisticRegression(class_weight='balanced', penalty='l1')],
        'classifier__C': [0.01,0.1,1,10,100],
    },
]

classifier = RandomizedSearchCV(estimator=BinaryRelevance(), param_distributions={parameters}, n_iter=15, cv=5, s
coring='f1_micro', n_jobs=-1, verbose=0)
classifier.fit(X_train_multilabel, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",classifier.best_params_)
print("Best Cross Validation Score: ",classifier.best_score_)
```

16.1.2 Fit the best estimator on the data

In [9]:

```
import warnings
warnings.filterwarnings("ignore")
from sklearn.problem_transform import BinaryRelevance

start = datetime.now()

classifier = BinaryRelevance(GaussianNB())

classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams66_binary.pkl')
```

Accuracy : 0.31231028667790894
Hamming loss 0.34772344013490725

Micro-average quality numbers
Precision: 0.4372, Recall: 0.5639, F1-measure: 0.4925

Macro-average quality numbers
Precision: 0.4227, Recall: 0.5377, F1-measure: 0.4709

Classification Report				
	precision	recall	f1-score	support
0	0.24	0.40	0.30	596
1	0.55	0.64	0.59	1155
2	0.48	0.58	0.52	911
micro avg	0.44	0.56	0.49	2662
macro avg	0.42	0.54	0.47	2662
weighted avg	0.46	0.56	0.50	2662
samples avg	0.29	0.33	0.29	2662

Time taken to run this cell : 0:04:54.669931

Out[9]:

['3_tags_char_ngrams66_binary.pkl']

17. ML Models with Classifier Chains + Taking average number of tags for each movie plots ~ 3

In the EDA section of analysis of tags, we have seen that there are almost 10500 movies which has tags less than or equal to 3.

In [5]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'split()' will tokenize each tag using space.
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

#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 3
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

17.1.1 Vectorize the plot synopsis using TFIDF Char 6 grams

In [6]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ngram_range=(6,6))
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 : 0:00:49.759399

In [18]:

```
import warnings
warnings.filterwarnings("ignore")

from skmultilearn.problem_transform import ClassifierChain
from sklearn.linear_model import LogisticRegression

start = datetime.now()

classifier = ClassifierChain(LogisticRegression(C=1))

classifier.fit(X_train_multilabel, y_train_multilabel)
predictions = classifier.predict(X_test_multilabel)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_char_ngrams66_clf_chains.pkl')
```

Accuracy : 0.4782462057335582
Hamming loss 0.2412591343451377

Micro-average quality numbers
Precision: 0.7122, Recall: 0.3253, F1-measure: 0.4466

Macro-average quality numbers
Precision: 0.6276, Recall: 0.2760, F1-measure: 0.3534

Classification Report

	precision	recall	f1-score	support
0	0.45	0.02	0.03	596
1	0.72	0.48	0.58	1155
2	0.70	0.33	0.45	911
micro avg	0.71	0.33	0.45	2662
macro avg	0.63	0.28	0.35	2662
weighted avg	0.66	0.33	0.41	2662
samples avg	0.21	0.19	0.19	2662

Time taken to run this cell : 0:00:43.695576

Out[18]:

['3_tags_char_ngrams66_clf_chains.pkl']

18. Using pre-trained google W2V models

In [3]:

```
#Load the processed dataset
dataframe=pd.read_csv("cleaned_movie_plots.csv")
dataframe.head()
```

Out[3]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming
0	0	\$	Set in Hamburg, West Germany, several criminal...	murder	test	set hamburg west germani sever crimin take adv...	set hamburg west germany several criminals tak...
1	1	\$windle	A 6th grader named Griffin Bing decides to gat...	flashback	train	grader name griffin bing decid gather entir gr...	grader named griffin bing decides gather entir...
2	2	'71	Gary Hook, a new recruit to the British Army, ...	suspenseful, neo noir, murder, violence	train	gari hook new recruit british armi take leav m...	gary hook new recruit british army takes leave...
3	3	'A' gai wak	Sergeant Dragon Ma (Jackie Chan) is part of th...	cult, violence	train	sergeant dragon jacki chan part hong kong mari...	sergeant dragon jackie chan part hong kong mar...
4	4	'Breaker' Morant	In Pretoria, South Africa, in 1902, Major Char...	murder, anti war, violence, flashback, tragedy...	train	pretoria south africa major charl bolton rod m...	pretoria south africa major charles bolton rod...

In [4]:

```
#Create a dataset for train and test
data_test=dataframe.loc[(dataframe['split'] == 'test')]
data_train=dataframe.loc[(dataframe['split'] == 'val') | (dataframe['split'] == 'train')]

#Split the whole data into train and test set
X_train = data_train['CleanedPlots_NoStemming']
y_train = data_train['tags']

X_test = data_test['CleanedPlots_NoStemming']
y_test = data_test['tags']

print("Number of points in training data: ",data_train.shape[0])
print("Number of points in test data: ",data_test.shape[0])
```

Number of points in training data: 11816
Number of points in test data: 2965

18.1 Loading Google W2V model

In [5]:

```
#https://drive.google.com/file/d/0B7XkCwpI5KDYnLnUTtLSS21pQmM/edit
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)
```

paramiko missing, opening SSH/SCP/SFTP paths will be disabled. `pip install paramiko` to suppress

18.2 Average Word2Vec using the Google W2V model.

In [6]:

```
#This method returns the Average Word2Vec vectors for all reviews in a given dataset
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 the averages at each iteration.
        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 [8:46:42<00:00, 1.72s/it]
100%|██████████| 2965/2965 [1:59:54<00:00, 1.88s/it]
```

19. ML models taking average number of tags for each movie plots ~ 3 + Average Word2Vec features

In the EDA section of analysis of tags, we have seen that there are almost 10500 movies which has tags less than or equal to 3.

In [5]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'split()' will tokenize each tag using space.
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

#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number = 3
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

19.1.0 Using Average W2V

In [27]:

```
import pickle

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)
```

19.1.1 Get best estimator using RandomSearch + Logistic Regression

In [17]:

```
st=datetime.now()
from scipy import stats

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:17:07.911899
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
                        fit_intercept=True, intercept_scaling=1, max_iter=100,
                        multi_class='warn', n_jobs=None, penalty='l1', random_state=None,
                        solver='warn', tol=0.0001, verbose=0, warm_start=False),
                        n_jobs=-1)
Best Cross Validation Score:  0.570961590564332
```

19.1.2 Fit the best estimator on the data

In [16]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train, y_train_multilabel)
predictions = classifier.predict(X_test)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_lr.pkl')
```

Accuracy : 0.3403035413153457
Hamming loss 0.3171444631815627

Micro-average quality numbers
Precision: 0.4797, Recall: 0.7040, F1-measure: 0.5706

Macro-average quality numbers
Precision: 0.4793, Recall: 0.6924, F1-measure: 0.5581

Classification Report				
	precision	recall	f1-score	support
0	0.28	0.62	0.39	596
1	0.63	0.72	0.67	1155
2	0.53	0.74	0.62	911
micro avg	0.48	0.70	0.57	2662
macro avg	0.48	0.69	0.56	2662
weighted avg	0.52	0.70	0.59	2662
samples avg	0.33	0.41	0.35	2662

Time taken to run this cell : 0:00:36.909625

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)

Out[16]:

['3_tags_avg_w2v.pkl']

19.2.1 Get best estimator using RandomSearch + Linear SVM

In [19]:

```
st=datetime.now()
from sklearn.svm import SVC

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha}

base_estimator = OneVsRestClassifier(SVC(kernel='linear',class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 1:18:17.921036
Best estimator: OneVsRestClassifier(estimator=SVC(C=1, cache_size=200, class_weight='balanced', coef0=0.0, decision_function_shape='ovr', degree=3, gamma='auto_deprecated', kernel='linear', max_iter=-1, probability=False, random_state=None, shrinking=True, tol=0.001, verbose=False), n_jobs=-1)
Best Cross Validation Score: 0.5760526066115443

19.2.2 Fit the best estimator on the data

In [20]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train, y_train_multilabel)
predictions = classifier.predict(X_test)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_xgb.pkl')
```

Accuracy : 0.3318718381112985
Hamming loss 0.3211916807195053

Micro-average quality numbers
Precision: 0.4757, Recall: 0.7168, F1-measure: 0.5719

Macro-average quality numbers
Precision: 0.4771, Recall: 0.7048, F1-measure: 0.5599

Classification Report

	precision	recall	f1-score	support
0	0.28	0.63	0.38	596
1	0.63	0.74	0.68	1155
2	0.53	0.74	0.62	911
micro avg	0.48	0.72	0.57	2662
macro avg	0.48	0.70	0.56	2662
weighted avg	0.52	0.72	0.59	2662
samples avg	0.33	0.42	0.35	2662

Time taken to run this cell : 0:02:27.623067

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

Out[20]:

```
['3_tags_avg_w2v_svm.pkl']
```

19.3.1 Get best estimator using RandomSearch + XGBoost Classifier

In [23]:

```
st=datetime.now()
from xgboost import XGBClassifier

params = {'estimator__learning_rate':stats.uniform(0.001,0.2),
          'estimator__n_estimators':[10,50,100,250,500,750,1000,2000],
          'estimator__gamma':stats.uniform(0,0.02),
          'estimator__subsample':(0.2,0.3,0.4,0.5,0.6, 0.7, 0.8),
          'estimator__reg_alpha':[25,50,75,100,150,200],
          'estimator__reg_lambda':[25,50,75,100,150,200],
          'estimator__max_depth':np.arange(1,11),
          'estimator__colsample_bytree':[0.2,0.3,0.4,0.5,0.6,0.7,0.8],
          'estimator__min_child_weight':np.arange(1,11)}

base_estimator = OneVsRestClassifier(XGBClassifier(), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=15, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning: 1:05:04.418486
Best estimator: OneVsRestClassifier(estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
colsample_bytree=0.6, gamma=0.012091859781317065,
learning_rate=0.09389519015780248, max_delta_step=0, max_depth=3,
min_child_weight=5, missing=None, n_estimators=750, n_jobs=1,
nthread=None, objective='binary:logistic', random_state=0,
reg_alpha=25, reg_lambda=75, scale_pos_weight=1, seed=None,
silent=True, subsample=0.5),
n_jobs=-1)
Best Cross Validation Score: 0.5299675562188216
```

19.3.2 Fit the best estimator on the data

In [24]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train, y_train_multilabel)
predictions = classifier.predict(X_test)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_xgb.pkl')
```

Accuracy : 0.4772344013490725
Hamming loss 0.23530073074761101

Micro-average quality numbers
Precision: 0.6630, Recall: 0.4346, F1-measure: 0.5251

Macro-average quality numbers
Precision: 0.6090, Recall: 0.3767, F1-measure: 0.4311

Classification Report

	precision	recall	f1-score	support
0	0.49	0.06	0.11	596
1	0.67	0.60	0.63	1155
2	0.67	0.47	0.55	911
micro avg	0.66	0.43	0.53	2662
macro avg	0.61	0.38	0.43	2662
weighted avg	0.63	0.43	0.49	2662
samples avg	0.29	0.25	0.26	2662

Time taken to run this cell : 0:02:01.719188

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

Out[24]:

```
['3_tags_avg_w2v_xgb.pkl']
```

19.4.1 Get best estimator using RandomSearch + RandomForest Classifier

In [10]:

```
st=datetime.now()
from sklearn.ensemble import RandomForestClassifier

params = {'estimator__n_estimators': [10,50,75,100,150,250,350,500,750,850,1000,1500,2000],
          'estimator__min_weight_fraction_leaf': [0,0.25,0.5],
          'estimator__max_depth': np.arange(1,6),
          'estimator__min_samples_leaf': np.arange(0.05,0.5,0.05),
          'estimator__min_samples_split': np.arange(0.05,1.0,0.05)}

base_estimator = OneVsRestClassifier(RandomForestClassifier(criterion='gini', class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=15, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:01:30.308574
Best estimator:  OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',
                                criterion='gini', max_depth=3, max_features='auto',
                                max_leaf_nodes=None, min_impurity_decrease=0.0,
                                min_impurity_split=None, min_samples_leaf=0.05,
                                min_samples_split=0.45, min_weight_fraction_leaf=0,
                                n_estimators=100, n_jobs=None, oob_score=False,
                                random_state=None, verbose=0, warm_start=False),
                                n_jobs=-1)
Best Cross Validation Score:  0.5400230088247128
```

19.4.2 Fit the best estimator on the data

In [12]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train, y_train_multilabel)
predictions = classifier.predict(X_test)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_rf.pkl')
```

Accuracy : 0.2701517706576729
Hamming loss 0.3563799887577291

Micro-average quality numbers
Precision: 0.4404, Recall: 0.7047, F1-measure: 0.5420

Macro-average quality numbers
Precision: 0.4374, Recall: 0.6900, F1-measure: 0.5284

Classification Report

	precision	recall	f1-score	support
0	0.26	0.59	0.36	596
1	0.59	0.73	0.65	1155
2	0.47	0.75	0.58	911
micro avg	0.44	0.70	0.54	2662
macro avg	0.44	0.69	0.53	2662
weighted avg	0.47	0.70	0.56	2662
samples avg	0.32	0.41	0.34	2662

Time taken to run this cell : 0:00:03.774659

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

Out[12]:

```
['3_tags_avg_w2v_rf.pkl']
```

20. Feature Engineering Section: Combining average Word2Vec features with TF-IDF char (6,6) grams features.

In [6]:

```
#Split the whole data into train and test set
X_train = data_train['CleanedPlots']
y_train = data_train['tags']

X_test = data_test['CleanedPlots']
y_test = data_test['tags']
```

In [8]:

```
#Importing & Initializing the "CountVectorizer" object, which is scikit-learn's bag of words tool. By default 'sp  
lit()' will tokenize each tag using space.  
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
```

In [10]:

```
start = datetime.now()  
  
#Take the maximum number of tags equal to the average number of tags as seen in the EDA section. Average number =  
3  
vectorizer = CountVectorizer(tokenizer = tokenize, binary='true', max_features=3).fit(y_train)  
y_train_multilabel = vectorizer.transform(y_train)  
y_test_multilabel = vectorizer.transform(y_test)  
  
#Use tf-idf vectorizer to vectorize the movie plot synopsis  
vectorizer = TfidfVectorizer(max_features=50000, strip_accents='unicode', analyzer='char', sublinear_tf=False, ng  
ram_range=(6,6))  
X_train_char6 = vectorizer.fit_transform(X_train)  
X_test_char6 = vectorizer.transform(X_test)  
  
#Load the average w2v features  
import pickle  
with open('X_train_W2V.pkl', 'rb') as file:  
    X_train_W2V = pickle.load(file)  
  
with open('X_test_W2V.pkl', 'rb') as file:  
    X_test_W2V = pickle.load(file)  
  
#Convert the dense matrix to sparse matrix  
from scipy import sparse  
X_train_W2V = sparse.csr_matrix(X_train_W2V)  
X_test_W2V = sparse.csr_matrix(X_test_W2V)  
  
#Stack the two sparse matrices into one single matrix  
from scipy.sparse import hstack  
X_train_combined = hstack((X_train_char6,X_train_W2V))  
X_test_combined = hstack((X_test_char6,X_test_W2V))  
  
print("Time taken to run this cell :", datetime.now() - start)
```

Time taken to run this cell : 0:00:45.801648

In []:

20.1.1 Get best estimator using RandomSearch + Logistic Regression

In [13]:

```
st=datetime.now()
from scipy import stats

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha,
          "estimator__penalty":penalty}

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_combined, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
Time taken to perform hyperparameter tuning:  0:17:31.458169
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced', dual=False,
                        fit_intercept=True, intercept_scaling=1, max_iter=100,
                        multi_class='warn', n_jobs=None, penalty='l2', random_state=None,
                        solver='warn', tol=0.0001, verbose=0, warm_start=False),
                        n_jobs=-1)
Best Cross Validation Score:  0.5871627511383386
```

20.1.2 Fit the best estimator on the data

In [15]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_combined, y_train_multilabel)
predictions = classifier.predict(X_test_combined)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_tfidf_lr.pkl')
```

Accuracy : 0.3935919055649241
Hamming loss 0.280831928049466

Micro-average quality numbers
Precision: 0.5248, Recall: 0.6529, F1-measure: 0.5819

Macro-average quality numbers
Precision: 0.5055, Recall: 0.6287, F1-measure: 0.5584

Classification Report

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

	precision	recall	f1-score	support
0	0.32	0.48	0.39	596
1	0.64	0.70	0.67	1155
2	0.55	0.70	0.62	911
micro avg	0.52	0.65	0.58	2662
macro avg	0.51	0.63	0.56	2662
weighted avg	0.54	0.65	0.59	2662
samples avg	0.33	0.38	0.33	2662

Time taken to run this cell : 0:00:22.352648

Out[15]:

```
['3_tags_avg_w2v_tfidf_lr.pkl']
```

20.2.1 Get best estimator using RandomSearch + Linear SVM

In [10]:

```
st=datetime.now()
from sklearn.svm import SVC

alpha = [0.000001,0.00001,0.0001,0.001,0.01,0.1,1,10,100,1000,10000]

penalty=['l1','l2']

params = {"estimator__C":alpha}

base_estimator = OneVsRestClassifier(SVC(kernel='linear',class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_combined, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

Time taken to perform hyperparameter tuning: 2:07:11.759412
Best estimator: OneVsRestClassifier(estimator=SVC(C=1, cache_size=200, class_weight='balanced', coe
f0=0.0,
decision_function_shape='ovr', degree=3, gamma='auto_deprecated',kernel='linear',
max_iter=-1, probability=False, random_state=None,
shrinking=True, tol=0.001, verbose=False),
n_jobs=-1)
Best Cross Validation Score: 0.5894572364661258

20.2.2 Fit the best estimator on the data

In [21]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_combined, y_train_multilabel)
predictions = classifier.predict(X_test_combined)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_tfidf_svc.pkl')
```

Accuracy : 0.5212547983651977
Hamming loss 0.2436697512465823

Micro-average quality numbers
Precision: 0.5344, Recall: 0.7368, F1-measure: 0.5983

Macro-average quality numbers
Precision: 0.4866, Recall: 0.7224, F1-measure: 0.5694

Classification Report

	precision	recall	f1-score	support
0	0.28	0.63	0.38	596
1	0.63	0.74	0.68	1155
2	0.53	0.74	0.62	911
micro avg	0.54	0.74	0.60	2662
macro avg	0.49	0.73	0.57	2662
weighted avg	0.53	0.73	0.59	2662
samples avg	0.42	0.51	0.52	2662

Time taken to run this cell : 0:03:05.645887

['3_tags_avg_w2v_tfidf_svc.pkl']

20.3.1 Get best estimator using RandomSearch + XGBoost Classifier

In [19]:

```
st=datetime.now()
from xgboost import XGBClassifier

params = {'estimator__learning_rate': stats.uniform(0.001,0.2),
          'estimator__n_estimators': [10,50,100,250,500,750,1000,2000],
          'estimator__gamma': stats.uniform(0,0.02),
          'estimator__subsample': (0.2,0.3,0.4,0.5,0.6, 0.7, 0.8),
          'estimator__reg_alpha': [25,50,75,100,150,200],
          'estimator__reg_lambda': [25,50,75,100,150,200],
          'estimator__max_depth': np.arange(1,11),
          'estimator__colsample_bytree': [0.2,0.3,0.4,0.5,0.6,0.7,0.8],
          'estimator__min_child_weight': np.arange(1,11)}

base_estimator = OneVsRestClassifier(XGBClassifier(), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=15, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_combined, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be caused by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

```
Time taken to perform hyperparameter tuning: 2:53:04.717010
Best estimator: OneVsRestClassifier(estimator=XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
  colsample_bytree=0.2, gamma=0.006669467455502056,
  learning_rate=0.04184653872859152, max_delta_step=0, max_depth=5,
  min_child_weight=2, missing=None, n_estimators=750, n_jobs=1,
  nthread=None, objective='binary:logistic', random_state=0,
  reg_alpha=25, reg_lambda=25, scale_pos_weight=1, seed=None,
  silent=True, subsample=0.5),
  n_jobs=-1)
Best Cross Validation Score: 0.5766368213315659
```

20.3.2 Fit the best estimator on the data

In [20]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_combined, y_train_multilabel)
predictions = classifier.predict(X_test_combined)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print (metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_tfidf_xgb.pkl')
```

Accuracy : 0.5018549747048904
Hamming loss 0.22102304665542438

Micro-average quality numbers
Precision: 0.6814, Recall: 0.4910, F1-measure: 0.5707

Macro-average quality numbers
Precision: 0.6687, Recall: 0.4397, F1-measure: 0.5040

Classification Report

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

	precision	recall	f1-score	support
0	0.64	0.17	0.27	596
1	0.69	0.65	0.67	1155
2	0.68	0.50	0.58	911
micro avg	0.68	0.49	0.57	2662
macro avg	0.67	0.44	0.50	2662
weighted avg	0.67	0.49	0.55	2662
samples avg	0.31	0.28	0.28	2662

Time taken to run this cell : 0:12:08.857904

Out[20]:

```
['3_tags_avg_w2v_tfidf_xgb.pkl']
```

20.4.1 Get best estimator using RandomSearch + RandomForest Classifier

In [16]:

```
st=datetime.now()
from sklearn.ensemble import RandomForestClassifier

params = {'estimator__n_estimators': [10,50,75,100,150,250,350,500,750,850,1000,1500,2000],
          'estimator__min_weight_fraction_leaf': [0,0.25,0.5],
          'estimator__max_depth': np.arange(1,6),
          'estimator__min_samples_leaf': np.arange(0.05,0.5,0.05),
          'estimator__min_samples_split': np.arange(0.05,1.0,0.05)}

base_estimator = OneVsRestClassifier(RandomForestClassifier(criterion='gini', class_weight='balanced'), n_jobs=-1)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=15, cv=5, scoring='f1_micro', n_jobs=-1, verbose=0)
rsearch_cv.fit(X_train_combined, y_train_multilabel)

print("Time taken to perform hyperparameter tuning: ",datetime.now()-st)
print("Best estimator: ",rsearch_cv.best_estimator_)
print("Best Cross Validation Score: ",rsearch_cv.best_score_)
```

```
/root/anaconda3/lib/python3.7/site-packages/sklearn/externals/joblib/externals/loky/process_executor
.py:706: UserWarning: A worker stopped while some jobs were given to the executor. This can be cause
d by a too short worker timeout or by a memory leak.
  "timeout or by a memory leak.", UserWarning
```

Time taken to perform hyperparameter tuning: 0:42:19.191889

Best estimator: OneVsRestClassifier(estimator=RandomForestClassifier(bootstrap=True, class_weight='balanced',

```
    criterion='gini', max_depth=5, max_features='auto',
    max_leaf_nodes=None, min_impurity_decrease=0.0,
    min_impurity_split=None, min_samples_leaf=0.05,
    min_samples_split=0.5, min_weight_fraction_leaf=0,
    n_estimators=750, n_jobs=None, oob_score=False,
    random_state=None, verbose=0, warm_start=False),
    n_jobs=-1)
```

Best Cross Validation Score: 0.5704428073527608

20.4.2 Fit the best estimator on the data

In [17]:

```
start = datetime.now()

classifier = rsearch_cv.best_estimator_
classifier.fit(X_train_combined, y_train_multilabel)
predictions = classifier.predict(X_test_combined)

print("Accuracy :",metrics.accuracy_score(y_test_multilabel, predictions))
print("Hamming loss ",metrics.hamming_loss(y_test_multilabel,predictions))

precision = precision_score(y_test_multilabel, predictions, average='micro')
recall = recall_score(y_test_multilabel, predictions, average='micro')
f1 = f1_score(y_test_multilabel, predictions, average='micro')

print("\nMicro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

precision = precision_score(y_test_multilabel, predictions, average='macro')
recall = recall_score(y_test_multilabel, predictions, average='macro')
f1 = f1_score(y_test_multilabel, predictions, average='macro')

print("\nMacro-average quality numbers")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))

print("\nClassification Report")
print(metrics.classification_report(y_test_multilabel, predictions))
print("Time taken to run this cell :", datetime.now() - start)

joblib.dump(classifier, '3_tags_avg_w2v_tfidf_rf.pkl')
```

Accuracy : 0.3497470489038786
Hamming loss 0.31197301854974707

Micro-average quality numbers
Precision: 0.4843, Recall: 0.6544, F1-measure: 0.5566

Macro-average quality numbers
Precision: 0.4720, Recall: 0.6312, F1-measure: 0.5361

Classification Report

	precision	recall	f1-score	support
0	0.28	0.50	0.36	596
1	0.63	0.71	0.67	1155
2	0.50	0.68	0.58	911
micro avg	0.48	0.65	0.56	2662
macro avg	0.47	0.63	0.54	2662
weighted avg	0.51	0.65	0.57	2662
samples avg	0.30	0.38	0.32	2662

Time taken to run this cell : 0:01:06.852654

/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1143: UndefinedMetricWarning: Precision and F-score are ill-defined and being set to 0.0 in samples with no predicted labels.

```
'precision', 'predicted', average, warn_for)
/root/anaconda3/lib/python3.7/site-packages/sklearn/metrics/classification.py:1145: UndefinedMetricWarning: Recall and F-score are ill-defined and being set to 0.0 in samples with no true labels.
'recall', 'true', average, warn_for)
```

Out[17]:

['3_tags_avg_w2v_rf.pkl']

Performance comparison of all the models

In [10]:

```
from IPython.core.display import display, HTML
display(HTML("<style>.container { width:100% !important; }</style>"))

from prettytable import PrettyTable

#Table 1
print("Baseline Models Word NGrams (Without Hyperparameter tuning): Section 8")
print("="*70)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF 1-Grams ', 0.0182,0.0807,0.2542,0.4562,0.3264])
table.add_row(["SGDClassifier + LogLoss", 'TF-IDF 1-Grams ', 0.0030,0.1347,0.1577,0.4937,0.2390])
table.add_row(["SGDClassifier + HingeLoss", 'TF-IDF 1-Grams ', 0.0033,0.1436,0.1437,0.4744,0.2206])
```

```

table.add_row(["SGDClassifier + HingeLoss", 'TF-IDF 1-1 Grams ', 0.0033,0.1430,0.1437,0.4777,0.2200])
table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams ', 0.0175,0.0884,0.2387,0.4863,0.3203])
table.add_row(["SGDClassifier + LogLoss", 'TF-IDF 1-2 Grams ', 0.0047,0.1361,0.1608,0.5160,0.2452])
table.add_row(["SGDClassifier + HingeLoss", 'TF-IDF 1-2 Grams ', 0.0030,0.1486,0.1476,0.5169,0.2296])
table.add_row(["LogisticRegression", 'TF-IDF 1-3 Grams ', 0.0172,0.0846,0.2457,0.4719,0.3232])
table.add_row(["SGDClassifier + HingeLoss", 'TF-IDF 1-3 Grams ', 0.0023,0.1465,0.1465,0.5014,0.2267])
table.add_row(["LogisticRegression", 'TF-IDF 1-4 Grams ', 0.0182,0.0846,0.2462,0.4725,0.3237])
table.add_row(["SGDClassifier + HingeLoss", 'TF-IDF 1-4 Grams ', 0.0033,0.1449,0.1466,0.4938,0.2260])

print(table)
print("=="*108+"\n"+"=="*108+"\n\n")

#Table 2
print("Word NGrams (With Hyperparameter tuning): Section 9 ")
print("=="*52)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss","Precision","Recall","Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF 1-1 Grams ', 0.0182,0.0807,0.2542,0.4562,0.3264])
table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams ', 0.0449,0.0628,0.3220,0.4212,0.3650])
table.add_row(["LogisticRegression", 'TF-IDF 1-3 Grams ', 0.0387,0.0651,0.3122,0.4319,0.3624])
table.add_row(["LogisticRegression", 'TF-IDF 1-4 Grams ', 0.0236,0.0831,0.2569,0.4965,0.3386])

print(table)
print("=="*100+"\n"+"=="*100+"\n\n")

#Table 3, Taking only 3 tags

#Table 3
print("Word NGrams ( Taking average number of tags for each movie plots = 3): Section 10")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss","Precision","Recall","Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Unigrams ', 0.3730,0.2980,0.5021,0.5045,0.5033])
table.add_row(["LogisticRegression", 'TF-IDF Bigrams ', 0.4357,0.2536,0.5978,0.4662,0.5238])
table.add_row(["LogisticRegression", 'TF-IDF Trigrams ', 0.3177,0.3326,0.4394,0.4046,0.4213])
table.add_row(["LogisticRegression", 'TF-IDF 4 Grams ', 0.3693,0.3234,0.3947,0.1514,0.2188])
table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams ', 0.3949,0.2848,0.5253,0.4996,0.5121])
table.add_row(["LogisticRegression", 'TF-IDF 1-3 Grams ', 0.3969,0.2845,0.5258,0.5015,0.5134])
table.add_row(["LogisticRegression", 'TF-IDF 1-4 Grams ', 0.3939,0.2857,0.5238,0.5000,0.5116])

print(table)
print("=="*100+"\n"+"=="*100+"\n\n")

#Table 4, Taking only 4 tags

#Table 4
print("Word NGrams ( Taking average number of tags for each movie plots = 4): Section 11")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss","Precision","Recall","Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Unigrams ', 0.2910,0.2743,0.4994,0.6202,0.5533])
table.add_row(["LogisticRegression", 'TF-IDF Bigrams ', 0.3399,0.2413,0.5739,0.4611,0.5114])
table.add_row(["LogisticRegression", 'TF-IDF Trigrams ', 0.2232,0.3182,0.4121,0.3789,0.3948])
table.add_row(["LogisticRegression", 'TF-IDF 4 Grams ', 0.2681,0.3022,0.3710,0.1487,0.2123])

table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams ', 0.3038,0.2702,0.5071,0.4838,0.4952])
table.add_row(["LogisticRegression", 'TF-IDF 1-3 Grams ', 0.3052,0.2695,0.5084,0.4857,0.4968])
table.add_row(["LogisticRegression", 'TF-IDF 1-4 Grams ', 0.3045,0.2699,0.5075,0.4888,0.4980])

print(table)
print("=="*100+"\n"+"=="*100+"\n\n")

#Table 5, Taking only 5 tags

#Table 5
print("Word NGrams ( Taking average number of tags for each movie plots = 5): Section 12")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss","Precision","Recall","Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Unigrams ', 0.2246,0.2762,0.4612,0.4613,0.5461])
table.add_row(["LogisticRegression", 'TF-IDF Bigrams ', 0.2738,0.2353,0.5593,0.3861,0.4568])
table.add_row(["LogisticRegression", 'TF-IDF Trigrams ', 0.1777,0.3025,0.3975,0.3500,0.3722])
table.add_row(["LogisticRegression", 'TF-IDF 4 Grams ', 0.2232,0.2877,0.3501,0.1432,0.2032])

table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams ', 0.2397,0.2638,0.4843,0.4561,0.4698])
table.add_row(["LogisticRegression", 'TF-IDF 1-3 Grams ', 0.2394,0.2636,0.4849,0.4576,0.4709])
table.add_row(["LogisticRegression", 'TF-IDF 1-4 Grams ', 0.2418,0.2640,0.4843,0.4671,0.4756])

print(table)
print("=="*100+"\n"+"=="*100+"\n\n")

#Table 6, Taking only 3 tags + Char Ngrams

```

#Table 6

```
print("Char NGrams ( Taking average number of tags for each movie plots = 3): Section 13")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Char Unigrams ', 0.1946, 0.4489, 0.3493, 0.5793, 0.4353])
table.add_row(["LogisticRegression", 'TF-IDF Char Bigrams ', 0.2937, 0.3548, 0.4368, 0.6416, 0.5198])
table.add_row(["LogisticRegression", 'TF-IDF Char Trigrams ', 0.3392, 0.3184, 0.4761, 0.6386, 0.5455])
table.add_row(["LogisticRegression", 'TF-IDF Char 4grams ', 0.3625, 0.3037, 0.4942, 0.6409, 0.5581])
table.add_row(["LogisticRegression", 'TF-IDF Char 5grams ', 0.3895, 0.2856, 0.5186, 0.6341, 0.5706])
table.add_row(["LogisticRegression", 'TF-IDF Char 6grams ', 0.3858, 0.2875, 0.5153, 0.6570, 0.5776])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-6grams ', 0.3824, 0.2896, 0.5128, 0.6491, 0.5729])
table.add_row(["LogisticRegression", 'TF-IDF Char 1-6grams ', 0.3672, 0.2992, 0.5000, 0.6011, 0.5459])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-4grams ', 0.3790, 0.2909, 0.5113, 0.6273, 0.5634])
```

```
print(table)
print("=="*105+"\n"+"=="*105+"\n\n")
```

#Table 7, Taking only 4 tags + Char Ngrams

#Table 7

```
print("Char NGrams ( Taking average number of tags for each movie plots = 4): Section 14")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Char Unigrams ', 0.0489, 0.5850, 0.3011, 0.8593, 0.4459])
table.add_row(["LogisticRegression", 'TF-IDF Char Bigrams ', 0.1966, 0.3473, 0.4135, 0.6402, 0.5025])
table.add_row(["LogisticRegression", 'TF-IDF Char Trigrams ', 0.2650, 0.2952, 0.4713, 0.6393, 0.5426])
table.add_row(["LogisticRegression", 'TF-IDF Char 4grams ', 0.2775, 0.2814, 0.4896, 0.6464, 0.5572])
table.add_row(["LogisticRegression", 'TF-IDF Char 5grams ', 0.2957, 0.2732, 0.5010, 0.6242, 0.5558])
table.add_row(["LogisticRegression", 'TF-IDF Char 6grams ', 0.2836, 0.2762, 0.4968, 0.6519, 0.5639])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-6grams ', 0.2317, 0.3101, 0.4552, 0.6713, 0.5425])
table.add_row(["LogisticRegression", 'TF-IDF Char 1-6grams ', 0.2650, 0.2906, 0.4778, 0.6559, 0.5529])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-4grams ', 0.2863, 0.2795, 0.4919, 0.6205, 0.5488])
```

```
print(table)
print("=="*105+"\n"+"=="*105+"\n\n")
```

#Table 8, Taking only 5 tags + Char Ngrams

#Table 8

```
print("Char NGrams ( Taking average number of tags for each movie plots = 5): Section 15")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF Char Unigrams ', 0.0748, 0.4389, 0.3107, 0.5845, 0.4057])
table.add_row(["LogisticRegression", 'TF-IDF Char Bigrams ', 0.1328, 0.3529, 0.3857, 0.5355, 0.4800])
table.add_row(["LogisticRegression", 'TF-IDF Char Trigrams ', 0.1986, 0.2959, 0.4446, 0.6211, 0.5182])
table.add_row(["LogisticRegression", 'TF-IDF Char 4grams ', 0.2236, 0.2752, 0.4711, 0.6003, 0.5279])
table.add_row(["LogisticRegression", 'TF-IDF Char 5grams ', 0.1993, 0.2855, 0.4590, 0.6392, 0.5343])
table.add_row(["LogisticRegression", 'TF-IDF Char 6grams ', 0.2242, 0.2723, 0.4754, 0.6047, 0.5323])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-6grams ', 0.2185, 0.2793, 0.4664, 0.6250, 0.5342])
table.add_row(["LogisticRegression", 'TF-IDF Char 1-6grams ', 0.2026, 0.2883, 0.4506, 0.5695, 0.5031])
table.add_row(["LogisticRegression", 'TF-IDF Char 3-4grams ', 0.1919, 0.2913, 0.4514, 0.6350, 0.5277])
```

```
print(table)
print("=="*105+"\n"+"=="*105+"\n\n")
```

#Table 9, Taking only 5 tags + Char Ngrams

#Table 9

```
print("Char NGrams ( Binary Relevance + Classifier Chains + Taking average number of tags for each movie plots = 3): Section 16 + 17")
print("=="*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]

table.add_row(["BinaryRelevance + Gaussian NB", 'TF-IDF Char 6grams ', 0.3123, 0.3477, 0.4372, 0.5639, 0.4925])
table.add_row(["ClassifierChains + LogisticRegression", 'TF-IDF Char 6grams ', 0.4784, 0.2412, 0.7122, 0.3253, 0.4446])
```

```
print(table)
print("=="*120+"\n"+"=="*120+"\n\n")
```

#Table 10, Taking only 3 tags + Average Word2Vec

#Table 8

```
print("Avg Word2Vec ( Taking average number of tags for each movie plots = 3): Section 19")
print("=="*82)
table = PrettyTable()
```

```

table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]

table.add_row(["LogisticRegression", 'Avg Word2Vec ', 0.3403, 0.3171, 0.4797, 0.7040, 0.5706])
table.add_row(["Linear SVM", 'TF-IDF Char Bigrams ', 0.3318, 0.3211, 0.4757, 0.7168, 0.5719])
table.add_row(["XGBoost Classifier", 'TF-IDF Char Trigrams ', 0.4772, 0.2353, 0.6630, 0.4346, 0.5251])
table.add_row(["Random Forest Classifier", 'TF-IDF Char 4grams ', 0.2701, 0.3563, 0.4404, 0.7047, 0.5421])
print(table)
print("=*110+\n"+="=*110+\n\n")

#Table 11, Taking only 3 tags + Average Word2Vec + TFIDF Char (6,6) Grams

#Table 11
print("Avg Word2Vec ( Taking average number of tags for each movie plots = 3): Section 20")
print("=*82)
table = PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss", "Precision", "Recall", "Micro F1"]
table.add_row(["LogisticRegression", 'Avg Word2Vec ', 0.3935, 0.2808, 0.5248, 0.6529, 0.5819])
table.add_row(["Linear SVM", 'TF-IDF Char Bigrams ', 0.5212, 0.2436, 0.5344, 0.7368, 0.5983])
table.add_row(["XGBoost Classifier", 'TF-IDF Char Trigrams ', 0.5018, 0.2210, 0.6814, 0.4910, 0.5707])
table.add_row(["Random Forest Classifier", 'TF-IDF Char 4grams ', 0.3497, 0.3119, 0.4843, 0.6544, 0.5566])
print(table)
print("=*110+\n"+="=*110+\n\n")

```

Baseline Models Word NGrams (Without Hyperparameter tuning): Section 8

Model	Vectorizer	Accuracy	Hamming loss	Precision	Recall	Micro F1
LogisticRegression	TF-IDF 1-Grams	0.0182	0.0807	0.2542	0.4562	0.3264
SGDClassifier + LogLoss	TF-IDF 1-Grams	0.003	0.1347	0.1577	0.4937	0.239
SGDClassifier + HingeLoss	TF-IDF 1-Grams	0.0033	0.1436	0.1437	0.4744	0.2206
LogisticRegression	TF-IDF 1-2 Grams	0.0175	0.0884	0.2387	0.4863	0.3203
SGDClassifier + LogLoss	TF-IDF 1-2 Grams	0.0047	0.1361	0.1608	0.516	0.2452
SGDClassifier + HingeLoss	TF-IDF 1-2 Grams	0.003	0.1486	0.1476	0.5169	0.2296
LogisticRegression	TF-IDF 1-3 Grams	0.0172	0.0846	0.2457	0.4719	0.3232
SGDClassifier + HingeLoss	TF-IDF 1-3 Grams	0.0023	0.1465	0.1465	0.5014	0.2267
LogisticRegression	TF-IDF 1-4 Grams	0.0182	0.0846	0.2462	0.4725	0.3237
SGDClassifier + HingeLoss	TF-IDF 1-4 Grams	0.0033	0.1449	0.1466	0.4938	0.226

Word NGrams (With Hyperparameter tuning): Section 9

Model	Vectorizer	Accuracy	Hamming loss	Precision	Recall	Micro F1
LogisticRegression	TF-IDF 1-1 Grams	0.0182	0.0807	0.2542	0.4562	0.3264
LogisticRegression	TF-IDF 1-2 Grams	0.0449	0.0628	0.322	0.4212	0.365
LogisticRegression	TF-IDF 1-3 Grams	0.0387	0.0651	0.3122	0.4319	0.3624
LogisticRegression	TF-IDF 1-4 Grams	0.0236	0.0831	0.2569	0.4965	0.3386

Word NGrams (Taking average number of tags for each movie plots = 3): Section 10

Model	Vectorizer	Accuracy	Hamming loss	Precision	Recall	Micro F1
LogisticRegression	TF-IDF Unigrams	0.373	0.298	0.5021	0.5045	0.5033
LogisticRegression	TF-IDF Bigrams	0.4357	0.2536	0.5978	0.4662	0.5238
LogisticRegression	TF-IDF Trigrams	0.3177	0.3326	0.4394	0.4046	0.4213
LogisticRegression	TF-IDF 4 Grams	0.3693	0.3234	0.3947	0.1514	0.2188
LogisticRegression	TF-IDF 1-2 Grams	0.3949	0.2848	0.5253	0.4996	0.5121

9	LogisticRegression	TF-IDF Char Unigrams	0.0489	0.585	0.3011	0.8593	0.445
5	LogisticRegression	TF-IDF Char Bigrams	0.1966	0.3473	0.4135	0.6402	0.502
6	LogisticRegression	TF-IDF Char Trigrams	0.265	0.2952	0.4713	0.6393	0.542
2	LogisticRegression	TF-IDF Char 4grams	0.2775	0.2814	0.4896	0.6464	0.557
8	LogisticRegression	TF-IDF Char 5grams	0.2957	0.2732	0.501	0.6242	0.555
9	LogisticRegression	TF-IDF Char 6grams	0.2836	0.2762	0.4968	0.6519	0.563
5	LogisticRegression	TF-IDF Char 3-6grams	0.2317	0.3101	0.4552	0.6713	0.542
9	LogisticRegression	TF-IDF Char 1-6grams	0.265	0.2906	0.4778	0.6559	0.552
8	LogisticRegression	TF-IDF Char 3-4grams	0.2863	0.2795	0.4919	0.6205	0.548
+-----+-----+-----+-----+-----+-----+-----+-----							
---+							
=====							
=====							
=====							
=====							

Model		Vectorizer	Accuracy	Hamming loss	Precision	Recall	Micro
7	LogisticRegression	TF-IDF Char Unigrams	0.0748	0.4389	0.3107	0.5845	0.405
9	LogisticRegression	TF-IDF Char Bigrams	0.1328	0.3529	0.3857	0.5355	0.48
2	LogisticRegression	TF-IDF Char Trigrams	0.1986	0.2959	0.4446	0.6211	0.518
3	LogisticRegression	TF-IDF Char 4grams	0.2236	0.2752	0.4711	0.6003	0.527
3	LogisticRegression	TF-IDF Char 5grams	0.1993	0.2855	0.459	0.6392	0.534
2	LogisticRegression	TF-IDF Char 6grams	0.2242	0.2723	0.4754	0.6047	0.532
1	LogisticRegression	TF-IDF Char 3-6grams	0.2185	0.2793	0.4664	0.625	0.534
7	LogisticRegression	TF-IDF Char 1-6grams	0.2026	0.2883	0.4506	0.5695	0.503
	LogisticRegression	TF-IDF Char 3-4grams	0.1919	0.2913	0.4514	0.635	0.527

Model		Vectorizer	Accuracy	Hamming loss	Precision
Recall	Micro F1				
BinaryRelevance + Gaussian NB		TF-IDF Char 6grams	0.3123	0.3477	0.4372
0.5639	0.4925				
ClassifierChains + LogisticRegression		TF-IDF Char 6grams	0.4784	0.2412	0.7122
0.3253	0.4446				

Model	Vectorizer	Accuracy	Hamming loss	Precision	Recall
LogisticRegression	Avg Word2Vec	0.3403	0.3171	0.4797	0.704
Linear SVM	TF-IDF Char Bigrams	0.3318	0.3211	0.4757	0.7168
XGBoost Classifier	TF-IDF Char Trigrams	0.4772	0.2353	0.663	0.4346
Random Forest Classifier	TF-IDF Char 4grams	0.2701	0.3563	0.4404	0.7047

Avg Word2Vec (Taking average number of tags for each movie plots = 3): Section 20

Model	Vectorizer	Accuracy	Hamming loss	Precision	Recall
LogisticRegression	Avg Word2Vec	0.3935	0.2808	0.5248	0.6529
Linear SVM	TF-IDF Char Bigrams	0.5212	0.2436	0.5344	0.7368
XGBoost Classifier	TF-IDF Char Trigrams	0.5018	0.221	0.6814	0.491
Random Forest Classifier	TF-IDF Char 4grams	0.3497	0.3119	0.4843	0.6544

What we did throughout this experiment:

The objective of this experiment was to suggest tags based on the movie plots collected from IMDB and Wikipedia.

The dataset was collected from <https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags> (<https://www.kaggle.com/cryptexcode/mpst-movie-plot-synopses-with-tags>). The given dataset contains the movie name associated with a movie ID. Each movie contains a summary of the plots about the movie and the tags column contains information about the tags associated with each of the movies. There are a total of 14,828 movies and we have 71 unique tags spread across the entire dataset. The 'split' column in the dataset contains information about how the data is to be splitted in train, test and cross validation dataset.

The problem that we have is a multi-label classification problem. Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A movie plot synopsis may either have tags like horror, sad, violence, brutal or it may have all of these 4 tags.

For building and evaluation of our machine learning models, we have chosen the micro averaged F1 score metric as our key performance indicator. It has been researched and found that micro averaged F1 score is the most ideal metric when we have a multi-label classification problem. As our secondary metrics we will also have weighted accuracy, hamming loss, weighted precision and weighted recall.

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 reaches its best 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 class imbalance.

'Macro f1 score':

Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.

<https://www.kaggle.com/wiki/MeanFScore> (<https://www.kaggle.com/wiki/MeanFScore>)

http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html (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://www.kaggle.com/wiki/HammingLoss> (<https://www.kaggle.com/wiki/HammingLoss>)

Data Loading Phase.

Since the dataset is given as a CSV file, we have used the popular pandas library to load the data. A very basic and high level information reveals that the dataset contains 14828 rows and 6 columns. The next thing we will do is to create an SQL DB from the given CSV file. This is done for ease of operation during the later stages of this Ipython notebook.

A basic distribution plot reveals that there are 9489 training samples, 2966 test samples and 2373 samples for cross validation. It is also observed that out of the total data that is provided to us, almost 28% of the movie plot synopsis is collected from IMDB and almost 72% of them are collected from Wikipedia.

In this experiment, we have used Random Grid Search algorithm to optimize our hyperparameters. Due to this, we will combine the training and cross validation data into a single training set and perform K fold cross validation on it. We will use the test data (which should be unseen by the model) to evaluate our models performance.

Checking for duplicate entries of rows.

A simple check reveals that there were 47 duplicate entries in the given dataset, that constitutes 0.32% of the entire data. We don't want these duplicate entries to affect our machine learning models in any way, hence we have removed them and created a new DB which doesn't contain any duplicated entries.

Checking the number of times each movie appeared in the dataset

On simple EDA, it revealed that there are 14743 movies which occurred only once in the dataset, 32 movies occurred twice, there were 4 movies which occurred 3 times and only 1 movie which occurred 5 times.

Checking for the distribution of tags per movie

There are as many as 5133 movies which contains just a single tag, 2990 movies contains 2 tags, 1924 movies contains 3 tags. The maximum number of tags present in a movie is 24. That's massive! The average number of tags per movie in the entire dataset was roughly equal to 3 (2.98 to be precise). On checking the countplot of the distribution of tags per movie, we have seen that the distribution is highly skewed towards the left. There are an extremely high number of movies which contains 5 or less tags and there were very very low number of movies which contains more than 5 tags. There are almost 550 movies which contains 10 or more tags.

Exploring the length of the movie plots

A high level statistics reveals that there the maximum length of movie plot summary was as high as 63959 characters and the lowest length being 442. The median length of all the movie plot synopsis consisted of 3825 characters. None of the movies synopsis contained any external reference, there was just one movie which contained html tags and almost 20 movies which contains greater than sign. Almost all the movies had punctuation marks and stopwords. Before building our machine learning models we have processed the dataset by removing stopwords, punctuations and also decontracted the occurrence of certain words like didn't to did not, shouldn't to should not etc.

A simple distribution plot revealed that the median value of the length of the title texts were somewhere around 15 and there are extremely few movies which had it's length of plot synopsis greater than 20000 characters.

Analysis of tags

There are a total of 71 unique tags present in the dataset. We have used a custom tokenize function which splits the tag data for each review based on 'comma' and also trim any whitespaces present before or after a tag occurs. Removing whitespaces was extremely important, or else it was giving the same tag twice - one with a whitespace and the other without it. For example without removing the whitespaces, 'absurd' and 'absurd' were considered two separate tags and we were getting a total of 140 odd tags. Only after we removed the whitespaces did we get the number of unique tags to be 71.

Then we created a new dataframe which contains two columns - the tag name and the number of times each of these tags occurred in the entire dataset. On sorting this dataframe in descending order, we see 5 of the highest occurring tags are - murder, violence, flashback, romantic and cult. Tags like revenge, psychedelic and comedy closely followed the top 5 tags. This proves that most audience likes to watch movies which are related to murder, violence etc, hence more and more movies are made on this subjects.

On plotting the distribution of the number of tags, we have also seen that almost 10 tags occurs more than 1000 times, almost 5 tags occurs more than 3000 times, 75% of the tags occurs less than 570 times and just 25% tags are present in the dataset which occurs more than 570 times.

We have also stored the tags which occurred more than 1000 times, 5000 times in separate lists.

Key Observations from the analysis of tags:

1. 75% of tags occurs less than 570 times across different movies.
2. 25% of tags occurs less than 119 times across different movies.
3. The maximum number of times a tag occurs in a movie is 5771
4. There are total 9 tags which are used more than 1000 times.
5. 1 tag is used more than 5000 times.
6. Most frequent tag (i.e. 'murder') is used 5771 times
7. Since some tags occur much more frequently than others, Micro-averaged F1-score is the appropriate metric for this problem.
8. Minimum number of tags in a movie plot is 1.
9. Maximum number of tags in a movie plot is 25
10. Average number of tags per movie was close to 3.
11. 10551 movies had tags less than or equal to 3.
12. 11789 movies had tags less than or equal to 4.
13. 12705 movies had tags less than or equal to 5.
14. 13331 movies had tags less than or equal to 6.

Word Clouds of tags

The word cloud of all the tags revealed the same thing - which of the tags occurred with maximum number of occurrences across the dataset. In a word cloud, words which appears the most has bigger font compared to the one which occurs less.

1. A look at the word cloud shows that "murder", "violence", "flashback", "romantic", "cult" are the most frequently occurring tags in the movie synopses plots.
2. There are lots of tags which occurs less frequently like - 'brainwashing', 'alternate history', 'queer', 'clever', 'claustrophobic', 'whimsical', 'feel-good', 'blaxploitation', 'western', 'grindhouse film', 'magical realism', 'suicidal', 'autobiographical', 'christian film', 'non fiction'

K-Means clustering on Bag of Words representation of tags

Here, we have used K means clustering to get an idea which of the tags have a tendency to occur together. We have initialized the clusters centroids using the K-Means++ algorithm and selected the optimal number of clusters using the elbow method.

Data cleaning stage

In this stage, we have used custom functions with regex to clean and pre-process the movie plots. The custom functions does the following:

1. Remove HTML tags if any.
2. Remove punctuations, alphanumeric words and special characters.
3. Remove words which have character length less than equal to 2.
4. Convert all the words to lower case letters to avoid duplication of same words in two caps.
5. Remove the stopwords present in the movies.
6. Stem the words in the movie synopsis plots.

Once this step is done, we will create a new dataset which will contain the clean texts. We will build our machine learning models based on top of the cleaned texts.

Building ML models.

We will take the train and cross validation data and merge them to create one single training dataset. We will use this training data for cross validation as well. We will test our models using various featurization techniques and finally evaluate our models performance on the test set.

Before, we proceed to build our ML models, we need to encode the given tags. We will use binary bag of word vectors to convert the tags into binary numbers. This will be our dependent variable and we will build our machine learning models which would predict a binary vector.

Featurizations

For all the models, we have used TFID features to vectorize the movie plot synopsis data. For the initial base models we have tried a simple Logistic Regression model, SGDClassifier with LogLoss and SGDClassifier with HingeLoss. We have used all these models with OneVsRestClassifier to build our ML models. We figured out that LogisticRegression seems to perform very well and achieved a greater micro average F1 score than the rest of the models. So we decided to stick to LogisticRegression for tuning our advance models.

For the baseline models, we have used all the 25 tags and a fixed value of the hyperparameter C/alpha. We have used various combinations of the word Ngram features. The different features we have used are TFIDF Unigrams, Bigrams, Trigrams and Ngrams. The best micro F1 score that we have obtained was 0.3264, which is at par with the performance of the models given in the actual research paper.

To build more powerful models, we have actually taken the top 3, 4 and 5 tags reselectively to binarize our tag vectors and used them as a predictor to build our ML models. We did this because we have found out that the average number of tags associated with each movies were close to 3 tags. Hence, logically it should give us a good performance boost if we build our models using the top 3 tags instead of all the tags.

For word Ngram features with top 3 tags, the best micro averaged F1 score obtained was 0.5238 with a weighted recall value of 0.5. This is a significant improvement from the baseline models.

For top 4 tags and word ngram features, the best micro averaged F1 score that we have obtained was 0.5533 with a weighted recall score of 0.62. This is a much more significant improvement than our previous base models.

The performance of our models dropped slightly when we actually took top 5 tags to vectorize our models. We have achieved a micro averaged F1 score of 0.5461 with a weighted recall score of 0.4613.

For more advance features we have used character Ngram features to build our model. Just like word ngrams, character ngram features are used to generate Unigram, Bigram, Trigram, 4grams, 5grams and 6Grams features. We have also used char ngrams featurizations of 1-6 char ngrams, 3-4 char ngrams and 2-6 char ngram features.

In the character ngrams featurization sections we have experimented and build our models using top 3, 4 and 5 tags.

For the top 3 tags features, we have obtained the best micro averaged f1 score with char 6 grams models. The best micro average f1 score improved slightly from the previous word ngrams models and is no at 0.5776 and there was also a significant improvement in the weighted recall values - 0.6570. That's a massive improvement from both the baseline models as well as the word ngram models.

For the top 4 tags models, the performance decreased slightly and the best micro averaged f1 score we got was close to 0.5639 with a weighted recall score of 0.6519.

The performance of the models starts to decrease significantly when we featurize the tags data with top 5 tags. The best weighted f1 score achieved in this case was 0.53 with a recall score of 0.62.

Conclusion:

The TFIDF char ngrams features, as described in the research paper proved to be a surprisingly powerful feature which improved both the weighted f1 as well as the recall score. The accuracy values also seems to have improved with the these features.

The maximum micro averaged F1 score we have obtained from the entire project is 0.5983 and the maximum value of weighted recall value that we have obtained is 0.74, which is a significant improvement from the models that were build in the actual research paper at <https://www.aclweb.org/anthology/L18-1274> (<https://www.aclweb.org/anthology/L18-1274>). The highest value of F1 score they have reached was 0.37 whereas the machine learning models we have built has reached a maximum weighted f1 score of 0.57. That seems like a massive improvement.

Future work:

Future versions of this work might include a many to many recurrent neural network to predict the tags, since we know recurrent neural networks are very robust in capturing sequential information. Also, we could further improve the models by adding more and more data to it. 14K data is very less to build a very powerful model. If we had more data and more computing power, I am sure we could actually get a very high micro averaged f1 score (more than 0.8).

References:

1. Research Paper: <https://www.aclweb.org/anthology/L18-1274> (<https://www.aclweb.org/anthology/L18-1274>)
2. Code References: <https://www.appliedaicourse.com/> (<https://www.appliedaicourse.com/>)
3. Ideas: https://en.wikipedia.org/wiki/Multi-label_classification (https://en.wikipedia.org/wiki/Multi-label_classification)

In [12]:

```
import pdfkit
pdfkit.from_file('MPST Movie Plot Synopses Tag Prediction.html', 'MPST Movie Plot Synopses Tag Prediction.pdf')
```

Loading page (1/2)
Warning: Failed to load file:///mnt/0AD801EDD801D7B9/AAIC Resume Projects: Actual/Movie Tags/custom.css (ignore)
Printing pages (2/2)
Done

Out[12]:

True