In [3]:

```
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

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

```
Autosaving every 180 seconds
```

In [2]:

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

Load the processed dataset

In [4]:

```
dataframe=pd.read_csv("cleaned_movie_plots.csv")
dataframe.shape
```

Out[4]:

```
(14781, 7)
```

In [5]:

```
dataframe["sent_to_words"]=dataframe["CleanedPlots"].apply(lambda x: x.split())
dataframe.head()
```

Out[5]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming	sent_to_words
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...	[set, hamburg, west, germani, sever, crimin, t...
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...	[grader, name, griffin, bing, decid, gather, e...
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...	[gari, hook, new, recruit, british, armi, take...
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...	[sergeant, dragon, jacki, chan, part, hong, ko...
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...	[pretoria, south, africa, major, charl, bolton...

In [6]:

```
data_words=[]
for sent in dataframe["sent_to_words"].values:
    data_words.append(sent)
```

Extract top 8 topics

In [7]:

```
# Create Dictionary Prior to topic modelling,
#we convert the tokenized and lemmatized text to a bag of words –
#which you can think of as a dictionary where the key is the word and value is the number of times that word occu
rs in the entire corpus.

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

id2word = corpora.Dictionary(data_words)

"""Now for each pre-processed document we use the dictionary object just created to convert that document into a
bag of words.
i.e for each document we create a dictionary reporting how many words and how many times those words appear."""
corpus = [id2word.doc2bow(text) for text in data_words]

# Build LDA model
lda_model = gensim.models.LdaMulticore(corpus=corpus,
                                       id2word=id2word,
                                       num_topics=8,
                                       random_state=100,
                                       chunksize=10,
                                       passes=10,
                                       alpha='symmetric',
                                       iterations=100,
                                       per_word_topics=True,
                                       workers=7)

pprint(lda_model.print_topics())
```

```
[(0,
  '0.006*"use" + 0.005*"one" + 0.004*"team" + 0.004*"kill" + 0.004*"ship" + '
  '0.004*"order" + 0.004*"escap" + 0.004*"forc" + 0.004*"take" + '
  '0.004*"destroy"'),
 (1,
  '0.011*"not" + 0.007*"love" + 0.006*"father" + 0.005*"famili" + 0.005*"one" '
  '+ 0.005*"time" + 0.005*"becom" + 0.005*"new" + 0.005*"day" + 0.005*"life"'),
 (2,
  '0.010*"kill" + 0.008*"king" + 0.007*"fight" + 0.005*"villag" + 0.005*"men" '
  '+ 0.005*"return" + 0.005*"son" + 0.005*"death" + 0.004*"armi" + '
  '0.004*"order"'),
 (3,
  '0.013*"tom" + 0.011*"get" + 0.007*"friend" + 0.006*"school" + 0.006*"jerri" '
  '+ 0.006*"chris" + 0.006*"danni" + 0.005*"tell" + 0.005*"jim" + '
  '0.005*"mike"'),
 (4,
  '0.018*"kill" + 0.012*"polic" + 0.008*"shoot" + 0.008*"john" + 0.008*"money" '
  '+ 0.007*"car" + 0.007*"men" + 0.007*"take" + 0.006*"gun" + 0.006*"get"'),
 (5,
  '0.013*"kill" + 0.009*"find" + 0.009*"jack" + 0.008*"hous" + 0.008*"murder" '
  '+ 0.008*"david" + 0.007*"harri" + 0.006*"polic" + 0.006*"mari" + '
  '0.006*"sam"'),
 (6,
  '0.010*"find" + 0.008*"back" + 0.006*"one" + 0.005*"see" + 0.005*"fall" + '
  '0.004*"tri" + 0.004*"away" + 0.004*"take" + 0.004*"run" + 0.004*"head"'),
 (7,
  '0.024*"tell" + 0.023*"not" + 0.018*"say" + 0.014*"see" + 0.014*"get" + '
  '0.013*"ask" + 0.011*"back" + 0.009*"look" + 0.008*"room" + 0.007*"come"')]
```

Get dominant topics for each movie plot summaries

In [8]:

```
#Code reference: https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
data = dataframe.CleanedPlots.values.tolist()

def format_topics_sentences(ldamodel=None, corpus=corpus, texts=data):
    # Init output
    sent_topics_df = pd.DataFrame()

    # Get main topic in each document
    for i, row_list in enumerate(ldamodel[corpus]):
        row = row_list[0] if ldamodel.per_word_topics else row_list
        # print(row)
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num), round(prop_topic,4), topic_keyw
ords]), ignore_index=True)
            else:
                break
    sent_topics_df.columns = ['Dominant_Topic', 'Perc_Contribution', 'Topic_Keywords']

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


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

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

Out[8]:

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	0	4.0	0.5667	kill, polic, shoot, john, money, car, men, tak...	[set, hamburg, west, germani, sever, crimin, t...
1	1	4.0	0.4100	kill, polic, shoot, john, money, car, men, tak...	[grader, name, griffin, bing, decid, gather, e...
2	2	2.0	0.3225	kill, king, fight, villag, men, return, son, d...	[gari, hook, new, recruit, british, armi, take...
3	3	0.0	0.3582	use, one, team, kill, ship, order, escap, forc...	[sergeant, dragon, jacki, chan, part, hong, ko...
4	4	2.0	0.4715	kill, king, fight, villag, men, return, son, d...	[pretoria, south, africa, major, charl, bolton...
5	5	4.0	0.4599	kill, polic, shoot, john, money, car, men, tak...	[custom, investig, cliff, holden, dean, jagger...
6	6	4.0	0.5057	kill, polic, shoot, john, money, car, men, tak...	[year, pass, sinc, event, crocodil, dunde, mic...
7	7	1.0	0.4777	not, love, father, famili, one, time, becom, n...	[local, auto, plant, fiction, hadleyvill, penn...
8	8	1.0	0.4884	not, love, father, famili, one, time, becom, n...	[inspector, glebski, arriv, hotel, dead, mount...
9	9	4.0	0.3469	kill, polic, shoot, john, money, car, men, tak...	[time, around, east, side, kid, gang, well, me...

In [9]:

```
#Save the top topics in a dataframe CSV file.
df_dominant_topic.to_csv("dominant_topics_8.csv")
```

Plot T-SNE for all the 8 topics

In [10]:

```
# Get topic weights and dominant topics
#Code reference: https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/
from sklearn.manifold import TSNE
from bokeh.plotting import figure, output_file, show
from bokeh.models import Label
from bokeh.io import output_notebook

# Get topic weights
topic_weights = []
for i, row_list in enumerate(lda_model[corpus]):
    topic_weights.append([w for i, w in row_list[0]])

# Array of topic weights
arr = pd.DataFrame(topic_weights).fillna(0).values

# Keep the well separated points (optional)
arr = arr[np.amax(arr, axis=1) > 0.35]

# Dominant topic number in each doc
topic_num = np.argmax(arr, axis=1)

# tSNE Dimension Reduction
tsne_model = TSNE(n_components=2, verbose=1, random_state=0, angle=.99, init='pca')
tsne_lda = tsne_model.fit_transform(arr)
```

```
[t-SNE] Computing 91 nearest neighbors...
[t-SNE] Indexed 10893 samples in 0.027s...
[t-SNE] Computed neighbors for 10893 samples in 0.776s...
[t-SNE] Computed conditional probabilities for sample 1000 / 10893
[t-SNE] Computed conditional probabilities for sample 2000 / 10893
[t-SNE] Computed conditional probabilities for sample 3000 / 10893
[t-SNE] Computed conditional probabilities for sample 4000 / 10893
[t-SNE] Computed conditional probabilities for sample 5000 / 10893
[t-SNE] Computed conditional probabilities for sample 6000 / 10893
[t-SNE] Computed conditional probabilities for sample 7000 / 10893
[t-SNE] Computed conditional probabilities for sample 8000 / 10893
[t-SNE] Computed conditional probabilities for sample 9000 / 10893
[t-SNE] Computed conditional probabilities for sample 10000 / 10893
[t-SNE] Computed conditional probabilities for sample 10893 / 10893
[t-SNE] Mean sigma: 0.052901
[t-SNE] KL divergence after 250 iterations with early exaggeration: 82.246628
[t-SNE] KL divergence after 1000 iterations: 1.625771
```

In [18]:

```
#Code reference: https://www.kaggle.com/konohayui/topic-modeling-on-quora-insincere-questions
import random

def generate_color():
    color = "#{:02x}{:02x}{:02x}".format(*map(lambda x: random.randint(0, 255), range(3)))
    return color

colormap = np.array([generate_color() for t in range(15)])
colormap

# Plot the Topic Clusters using Bokeh
output_notebook()
n_topics = 8
mycolors = np.array([color for color in colormap])
plot = figure(title="t-SNE Clustering of {} LDA Topics".format(n_topics),
              plot_width=900, plot_height=700)
plot.scatter(x=tsne_lda[:,0], y=tsne_lda[:,1], color=mycolors[topic_num])
show(plot)
```

(https://bokeh.pydata.org) BokehJS 1.0.2 successfully loaded.

1. Train model with TFIDF 1-2 Grams + Top 8 Topics

In [18]:

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

#Load the topics data
df_topic=pd.read_csv("dominant_topics_8.csv")

#Combine the processed db with topics db
combined_df=pd.concat([dataframe,df_topic], axis=1)
combined_df.head()
```

Out[18]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming	Unnamed: 0	Document_No
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...	0	0
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...	1	1
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...	2	2
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...	3	3
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...	4	4

In [19]:

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

#Split the whole plot data into train and test set
X_train_plots = data_train['CleanedPlots']
X_test_plots = data_test['CleanedPlots']

#Split the whole topics data into train and test set
X_train_topics = data_train['Keywords']
X_test_topics = data_test['Keywords']

#Split the whole topics data into train and test set
X_train_cont = data_train['Topic_Perc_Contrib']
X_test_cont = data_test['Topic_Perc_Contrib']

#Split the tags
y_train = data_train['tags']
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
```

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
```

Vectorize the plots with TF-IDF 1-2 Grams + Binarize the tags

In [6]:

```
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_plots_multilabel = vectorizer.fit_transform(X_train_plots)
X_test_plots_multilabel = vectorizer.transform(X_test_plots)

#Use tf-idf vectorizer to vectorize the movie plot synopsis
vectorizer = TfidfVectorizer( smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), sublinear_tf=False,
ngram_range=(1,1))
X_train_topics_multilabel = vectorizer.fit_transform(X_train_topics)
X_test_topics_multilabel = vectorizer.transform(X_test_topics)

#Combine the sparse matrices into a single sparse matrix
from scipy.sparse import hstack
X_train_multilabel=hstack([X_train_plots_multilabel,X_train_topics_multilabel])
X_test_multilabel=hstack([X_test_plots_multilabel,X_test_topics_multilabel])

#Convert the tags to binary vectors
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 : 0:00:28.178249
```

In [7]:

```
print(X_train_multilabel.shape)
print(X_test_multilabel.shape)

print(y_train_multilabel.shape)
print(y_test_multilabel.shape)

del(dataframe,df_topic,combined_df,data_test,data_train,X_train_plots,X_test_plots,X_train_topics,X_test_topics,y
_train,y_test)
import gc
gc.collect()
```

```
(11816, 100065)
(2965, 100065)
(11816, 71)
(2965, 71)
```

Out[7]:

```
11
```

Get best estimator using RandomSearch + Logistic Regression (Since this was our previous best performing model)

In [20]:

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

st=datetime.now()

alpha = [0.001,0.01,0.1,0.5,0.9,1,1.5,10,100,1000]
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:  0:49:57.381819
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1.5, class_weight='balanced', du
al=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:  0.36308143924382335
```

Fit the best estimator on the data

In [23]:

```
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_Topic_Modelling_V1.pkl')
```

```
Accuracy : 0.045531197301854974
Hamming loss  0.0625941144336508

Micro-average quality numbers
Precision: 0.3231, Recall: 0.4206, F1-measure: 0.3684

Macro-average quality numbers
Precision: 0.1667, Recall: 0.2163, F1-measure: 0.1837

Classification Report
              precision    recall  f1-score   support

           0       0.16      0.11      0.13        56
           1       0.20      0.40      0.27       129
           2       0.17      0.18      0.18        28
           3       0.18      0.14      0.15        22
           4       0.30      0.50      0.37        18
           5       0.07      0.09      0.08        35
```

6	0.22	0.27	0.24	30
7	0.09	0.13	0.11	79
8	0.00	0.00	0.00	8
9	0.08	0.07	0.07	45
10	0.10	0.09	0.10	11
11	0.04	0.03	0.03	40
12	0.10	0.12	0.11	115
13	0.07	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	0.27	0.38	0.31	368
18	0.17	0.19	0.18	27
19	0.11	0.10	0.11	97
20	0.35	0.47	0.40	551
21	0.06	0.07	0.07	41
22	0.11	0.13	0.12	85
23	0.13	0.10	0.11	42
24	0.10	0.13	0.11	83
25	0.20	0.28	0.23	159
26	0.39	0.42	0.41	112
27	0.00	0.00	0.00	11
28	0.30	0.41	0.35	596
29	0.33	0.55	0.41	190
30	0.31	0.57	0.40	83
31	0.13	0.17	0.15	12
32	0.21	0.38	0.27	26
33	0.16	0.20	0.18	64
34	0.17	0.28	0.22	25
35	0.05	0.03	0.04	29
36	0.24	0.55	0.34	92
37	0.18	0.23	0.20	172
38	0.20	0.24	0.22	136
39	0.21	0.09	0.13	32
40	0.13	0.07	0.10	40
41	0.00	0.00	0.00	5
42	0.09	0.13	0.11	93
43	0.64	0.68	0.66	1155
44	0.11	0.16	0.13	117
45	0.23	0.52	0.32	145
46	0.00	0.00	0.00	5
47	0.24	0.33	0.28	129
48	0.11	0.11	0.11	36
49	0.12	0.11	0.12	44
50	0.19	0.14	0.16	28
51	0.17	0.18	0.17	51
52	0.34	0.43	0.38	395
53	0.12	0.12	0.12	65
54	0.00	0.00	0.00	15
55	0.07	0.05	0.06	37
56	0.32	0.45	0.37	507
57	0.42	0.59	0.49	587
58	0.20	0.24	0.22	148
59	0.25	0.28	0.26	173
60	0.27	0.56	0.36	66
61	0.13	0.14	0.14	51
62	0.05	0.05	0.05	66
63	0.05	0.05	0.05	39
64	0.00	0.00	0.00	8
65	0.22	0.46	0.30	228
66	0.13	0.07	0.10	27
67	0.10	0.11	0.11	119
68	0.55	0.68	0.61	911
69	0.30	0.43	0.35	14
70	0.00	0.00	0.00	13
micro avg	0.32	0.42	0.37	9021
macro avg	0.17	0.22	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 : 0:01:32.350270
```

```
['LR_Topic_Modelling_V1.pkl']
```

2. Using Glove vectors to vectorize the movie plots + Logistic Regression

2.1 Extract glove vectors for summaries

In [2]:

```
#http://nlp.stanford.edu/data/glove.6B.zip
glove=Glove()
model=glove.load_stanford("glove/glove.6B.300d.txt")
vocab=list(model.dictionary.keys())
```

In [2]:

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

Out[2]:

	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 [3]:

```
#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
```

In [5]:

```
#This method returns the Average Word2Vec vectors for all reviews in a given dataset
def vectorize_glove(dataset, word2vec_model, vocab):
    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 ve
ctors 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 vocab:
                word_vectors = model.word_vectors[model.dictionary[word]] #Creating a vector(numpy array of 300 d
imensions) 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_glove(X_train, model, vocab)
X_test_vectors = vectorize_glove(X_test, model, vocab)

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

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

```
100%|██████████| 11816/11816 [1:27:46<00:00,  3.59it/s]
100%|██████████| 2965/2965 [21:57<00:00,  2.44it/s]
```

2.2 Extract glove vectors for topics

In [4]:

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

#Load the topics data
df_topic=pd.read_csv("dominant_topics_8.csv")

#Combine the processed db with topics db
combined_df=pd.concat([dataframe,df_topic], axis=1)
combined_df.head(5)
```

Out[4]:

	index	title	plot_synopsis	tags	split	CleanedPlots	CleanedPlots_NoStemming	Unnamed: 0	Document_No
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...	0	0
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...	1	1
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...	2	2
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...	3	3
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...	4	4

In [13]:

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

#Split the whole topics data into train and test set
X_train_topics = data_train['Keywords']
X_test_topics = data_test['Keywords']

#Split the tags
y_train = data_train['tags']
y_test = data_test['tags']

X_train_topic_vectors = vectorize_glove(X_train_topics, model, vocab)
X_test_topic_vectors = vectorize_glove(X_test_topics, model, vocab)

with open('X_train_topic_vectors.pkl', 'wb') as file:
    pickle.dump(X_train_topic_vectors, file)

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

```
100%|██████████| 11816/11816 [43:23<00:00,  4.26it/s]
100%|██████████| 2965/2965 [10:49<00:00,  4.62it/s]
```

2.3 Load the glove vectors

In [5]:

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

#Split the whole topics data into train and test set
X_train_topics = data_train['Keywords']
X_test_topics = data_test['Keywords']

import pickle

with open('X_train_glove.pkl', 'rb') as file:
    X_train_glove = pickle.load(file)

with open('X_test_glove.pkl', 'rb') as file:
    X_test_glove = pickle.load(file)

#Split the tags
y_train = data_train['tags']
y_test = data_test['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
```

In [7]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the topic keywords
vectorizer = TfidfVectorizer(smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(","), sublinear_tf=False,
ngram_range=(1,1))
X_train_topics_multilabel = vectorizer.fit_transform(X_train_topics)
X_test_topics_multilabel = vectorizer.transform(X_test_topics)

#Combine the sparse matrices into a single sparse matrix
#Here we are combining the glove vectors matrix and the tf-idf topic keywords matrix
from scipy.sparse import hstack
from scipy import sparse
X_train_multilabel=hstack([sparse.csr_matrix(X_train_glove),X_train_topics_multilabel])
X_test_multilabel=hstack([sparse.csr_matrix(X_test_glove),X_test_topics_multilabel])

#Convert the tags to binary vectors
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 : 0:00:00.347635
```

2.4 Hyper parameter tuining using RandomSearch + Logistic Regression

In [8]:

```
from sklearn.model_selection import RandomizedSearchCV
from scipy import stats
warnings.filterwarnings("ignore")

st=datetime.now()

alpha = [0.001,0.01,0.1,0.5,0.9,1,1.5,10,100,1000,5000,10000]
penalty=['l1','l2']

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

base_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=6)
rsearch_cv = RandomizedSearchCV(estimator=base_estimator, param_distributions=params, n_iter=10, cv=5, scoring='f
1_micro', n_jobs=6, verbose=2)
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_)
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
[CV] estimator__penalty=l2, estimator__C=1.5 .....................
[CV] estimator__penalty=l2, estimator__C=1.5 .....................
[CV] estimator__penalty=l2, estimator__C=1.5 .....................
[CV] estimator__penalty=l2, estimator__C=1.5 .....................
[CV] estimator__penalty=l2, estimator__C=1.5 .....................
[CV] estimator__penalty=l1, estimator__C=0.001 ...................
[CV] ........ estimator__penalty=l1, estimator__C=0.001, total=  11.0s
[CV] estimator__penalty=l1, estimator__C=0.001 ...................
[CV] ........ estimator__penalty=l1, estimator__C=0.001, total=  11.4s
[CV] estimator__penalty=l1, estimator__C=0.001 ...................
[CV] ........ estimator__penalty=l1, estimator__C=0.001, total=  10.8s
[CV] estimator__penalty=l1, estimator__C=0.001 ...................
[CV] ........ estimator__penalty=l1, estimator__C=0.001, total=  10.8s
[CV] estimator__penalty=l1, estimator__C=0.001 ...................
[CV] ........ estimator__penalty=l1, estimator__C=0.001, total=  11.3s
[CV] estimator__penalty=l2, estimator__C=0.5 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.5, total= 3.9min
[CV] estimator__penalty=l2, estimator__C=0.5 .....................
[CV] .......... estimator__penalty=l2, estimator__C=1.5, total= 5.0min
[CV] estimator__penalty=l2, estimator__C=0.5 .....................
[CV] .......... estimator__penalty=l2, estimator__C=1.5, total= 5.0min
[CV] estimator__penalty=l2, estimator__C=0.5 .....................
[CV] .......... estimator__penalty=l2, estimator__C=1.5, total= 5.1min
[CV] estimator__penalty=l2, estimator__C=0.5 .....................
[CV] .......... estimator__penalty=l2, estimator__C=1.5, total= 5.1min
[CV] estimator__penalty=l1, estimator__C=10 ......................
[CV] .......... estimator__penalty=l2, estimator__C=1.5, total= 5.3min
[CV] estimator__penalty=l1, estimator__C=10 ......................
[CV] .......... estimator__penalty=l2, estimator__C=0.5, total= 3.6min
[CV] estimator__penalty=l1, estimator__C=10 ......................
[CV] .......... estimator__penalty=l2, estimator__C=0.5, total= 3.6min
[CV] estimator__penalty=l1, estimator__C=10 ......................
[CV] .......... estimator__penalty=l2, estimator__C=0.5, total= 3.6min
[CV] estimator__penalty=l1, estimator__C=10 ......................
[CV] .......... estimator__penalty=l2, estimator__C=0.5, total= 3.7min
[CV] estimator__penalty=l1, estimator__C=1000 ....................
[CV] .......... estimator__penalty=l1, estimator__C=10, total=484.2min
[CV] estimator__penalty=l1, estimator__C=1000 ....................
[CV] .......... estimator__penalty=l1, estimator__C=10, total=500.6min
[CV] estimator__penalty=l1, estimator__C=1000 ....................
[CV] .......... estimator__penalty=l1, estimator__C=10, total=508.4min
[CV] estimator__penalty=l1, estimator__C=1000 ....................
[CV] .......... estimator__penalty=l1, estimator__C=10, total=508.1min
[CV] estimator__penalty=l1, estimator__C=1000 ....................
[CV] .......... estimator__penalty=l1, estimator__C=10, total=522.5min
[CV] estimator__penalty=l2, estimator__C=0.9 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.9, total= 3.3min
[CV] estimator__penalty=l2, estimator__C=0.9 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.9, total= 3.4min
[CV] estimator__penalty=l2, estimator__C=0.9 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.9, total= 3.4min
[CV] estimator__penalty=l2, estimator__C=0.9 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.9, total= 3.4min
[CV] estimator__penalty=l2, estimator__C=0.9 .....................
[CV] .......... estimator__penalty=l2, estimator__C=0.9, total= 3.3min
[CV] estimator__penalty=l2, estimator__C=0.01 ....................
[CV] ......... estimator__penalty=l2, estimator__C=0.01, total= 1.0min
[CV] estimator__penalty=l2, estimator__C=0.01 ....................
[CV] ......... estimator__penalty=l2, estimator__C=0.01, total= 1.0min
[CV] estimator__penalty=l2, estimator__C=0.01 ....................
[CV] ......... estimator__penalty=l2, estimator__C=0.01, total= 1.0min
[CV] estimator__penalty=l2, estimator__C=0.01 ....................
[CV] ......... estimator__penalty=l2, estimator__C=0.01, total=  59.7s
[CV] estimator__penalty=l2, estimator__C=0.01 ....................

[Parallel(n_jobs=6)]: Done  29 tasks      | elapsed: 548.8min
```

```
[CV] ......... estimator__penalty=l2, estimator__C=0.01, total= 1.0min
[CV] estimator__penalty=l1, estimator__C=5000 ........................
[CV] ........ estimator__penalty=l1, estimator__C=1000, total=811.0min
[CV] estimator__penalty=l1, estimator__C=5000 ........................
[CV] ........ estimator__penalty=l1, estimator__C=5000, total=679.8min
[CV] estimator__penalty=l1, estimator__C=5000 ........................
[CV] ........ estimator__penalty=l1, estimator__C=1000, total=787.3min
[CV] estimator__penalty=l1, estimator__C=5000 ........................
[CV] ........ estimator__penalty=l1, estimator__C=1000, total=801.4min
[CV] estimator__penalty=l1, estimator__C=5000 ........................
[CV] ........ estimator__penalty=l1, estimator__C=1000, total=831.3min
[CV] estimator__penalty=l1, estimator__C=0.1 .........................
[CV] ........ estimator__penalty=l1, estimator__C=1000, total=832.0min
[CV] estimator__penalty=l1, estimator__C=0.1 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.1, total= 6.1min
[CV] estimator__penalty=l1, estimator__C=0.1 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.1, total= 6.1min
[CV] estimator__penalty=l1, estimator__C=0.1 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.1, total= 6.7min
[CV] estimator__penalty=l1, estimator__C=0.1 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.1, total= 6.1min
[CV] estimator__penalty=l1, estimator__C=0.5 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.1, total= 6.5min
[CV] estimator__penalty=l1, estimator__C=0.5 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.5, total=45.2min
[CV] estimator__penalty=l1, estimator__C=0.5 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.5, total=46.9min
[CV] estimator__penalty=l1, estimator__C=0.5 .........................
[CV] .......... estimator__penalty=l1, estimator__C=0.5, total=41.7min
[CV] estimator__penalty=l1, estimator__C=0.5 .........................
[CV] ........ estimator__penalty=l1, estimator__C=5000, total=628.7min
[CV] .......... estimator__penalty=l1, estimator__C=0.5, total=44.8min
[CV] .......... estimator__penalty=l1, estimator__C=0.5, total=30.9min
[CV] ........ estimator__penalty=l1, estimator__C=5000, total=462.9min
[CV] ........ estimator__penalty=l1, estimator__C=5000, total=435.2min
[CV] ........ estimator__penalty=l1, estimator__C=5000, total=405.9min

[Parallel(n_jobs=6)]: Done  50 out of  50 | elapsed: 1724.2min finished

Time taken to perform hyperparameter tuning:  1 day, 7:27:55.561498
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1000, class_weight='balanced', d
ual=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=6)
Best Cross Validation Score:  0.2360374972305146
```

2.5 Fit the best estimator on the data

In [9]:

```
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_Topic_Modelling_GLOVE_ALL_71_Tags.pkl')
```

```
Accuracy : 0.0003372681281618887
```

Hamming loss 0.16729449207894925

Micro-average quality numbers
Precision: 0.1471, Recall: 0.6055, F1-measure: 0.2367

Macro-average quality numbers
Precision: 0.0923, Recall: 0.4127, F1-measure: 0.1395

Classification Report

	precision	recall	f1-score	support
0	0.04	0.41	0.07	56
1	0.13	0.67	0.22	129
2	0.06	0.39	0.11	28
3	0.02	0.32	0.03	22
4	0.05	0.44	0.08	18
5	0.03	0.34	0.05	35
6	0.09	0.43	0.15	30
7	0.05	0.43	0.09	79
8	0.00	0.00	0.00	8
9	0.04	0.42	0.07	45
10	0.06	0.18	0.09	11
11	0.03	0.38	0.05	40
12	0.05	0.42	0.10	115
13	0.01	0.17	0.02	18
14	0.03	0.11	0.05	9
15	0.01	0.07	0.02	15
16	0.00	0.00	0.00	13
17	0.21	0.61	0.31	368
18	0.04	0.19	0.06	27
19	0.07	0.53	0.12	97
20	0.29	0.63	0.40	551
21	0.04	0.44	0.08	41
22	0.06	0.51	0.10	85
23	0.03	0.38	0.06	42
24	0.06	0.49	0.11	83
25	0.10	0.55	0.17	159
26	0.14	0.62	0.22	112
27	0.02	0.09	0.03	11
28	0.28	0.57	0.37	596
29	0.19	0.72	0.30	190
30	0.11	0.75	0.20	83
31	0.02	0.08	0.04	12
32	0.06	0.46	0.11	26
33	0.07	0.48	0.13	64
34	0.04	0.40	0.07	25
35	0.02	0.28	0.04	29
36	0.12	0.77	0.21	92
37	0.11	0.61	0.18	172
38	0.09	0.51	0.15	136
39	0.05	0.22	0.08	32
40	0.03	0.28	0.05	40
41	0.00	0.00	0.00	5
42	0.08	0.60	0.14	93
43	0.62	0.73	0.67	1155
44	0.09	0.52	0.15	117
45	0.15	0.75	0.25	145
46	0.00	0.00	0.00	5
47	0.13	0.65	0.22	129
48	0.02	0.28	0.04	36
49	0.03	0.45	0.06	44
50	0.05	0.46	0.10	28
51	0.04	0.43	0.08	51
52	0.27	0.67	0.38	395
53	0.04	0.37	0.08	65
54	0.03	0.20	0.05	15
55	0.01	0.19	0.03	37
56	0.30	0.66	0.42	507
57	0.38	0.66	0.48	587
58	0.11	0.58	0.18	148
59	0.13	0.61	0.21	173
60	0.11	0.68	0.19	66
61	0.03	0.33	0.06	51
62	0.04	0.42	0.07	66
63	0.01	0.18	0.02	39
64	0.00	0.00	0.00	8
65	0.17	0.68	0.27	228
66	0.03	0.26	0.06	27
67	0.07	0.50	0.12	119
68	0.52	0.72	0.60	911
69	0.13	0.29	0.18	14
70	0.01	0.08	0.02	13

```
avg / total       0.27      0.61      0.35      9021

Time taken to run this cell : 3:13:14.055507
```

Out[9]:

```
['LR_Topic_Modelling_GLOVE_ALL_71_Tags.pkl']
```

In [ ]:

```
```

3. 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
```

3.1 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
```

3.2 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 ve
ctors 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]
```

3.3. ML models taking number of tags = 71 + 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 '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').fit(y_train)
y_train_multilabel = vectorizer.transform(y_train)
y_test_multilabel = vectorizer.transform(y_test)
```

3.4 Using Average W2V

In [12]:

```
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)
```

In [28]:

```
start = datetime.now()

#Use tf-idf vectorizer to vectorize the topic keywords
vectorizer = TfidfVectorizer(smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(","), sublinear_tf=False,
ngram_range=(1,1))
X_train_topics_multilabel = vectorizer.fit_transform(X_train_topics)
X_test_topics_multilabel = vectorizer.transform(X_test_topics)

#Combine the sparse matrices into a single sparse matrix
#Here we are combining the avg w2v vectors matrix and the tf-idf topic keywords matrix
X_train_multilabel=hstack([sparse.csr_matrix(X_train),X_train_topics_multilabel])
X_test_multilabel=hstack([sparse.csr_matrix(X_test),X_test_topics_multilabel])

#Convert the tags to binary vectors
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 : 0:00:00.754996
```

3.5 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='f
1_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:  17:00:14.993733
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='l1', random_state=None,
          solver='liblinear', tol=0.0001, verbose=0, warm_start=False),
          n_jobs=-1)
Best Cross Validation Score:  0.2358312952138803
```

3.6 Fit the best estimator on the data

In [4]:

```
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, '71_tags_avg_w2v_lr.pkl')
```

```
Accuracy : 0.0003372681281618887
Hamming loss  0.16742274897275727

Micro-average quality numbers
Precision: 0.1483, Recall: 0.6128, F1-measure: 0.2388

Macro-average quality numbers
Precision: 0.0931, Recall: 0.4304, F1-measure: 0.1412

Classification Report
              precision    recall  f1-score   support

           0       0.04      0.50      0.08        56
           1       0.14      0.69      0.23       129
           2       0.04      0.29      0.08        28
           3       0.02      0.32      0.03        22
           4       0.03      0.28      0.05        18
           5       0.04      0.60      0.08        35
           6       0.06      0.33      0.10        30
           7       0.05      0.42      0.09        79
           8       0.00      0.00      0.00         8
           9       0.04      0.44      0.08        45
          10       0.09      0.36      0.14        11
          11       0.02      0.35      0.04        40
          12       0.06      0.48      0.11       115
          13       0.02      0.28      0.04        18
          14       0.00      0.00      0.00         9
          15       0.00      0.00      0.00        15
          16       0.02      0.38      0.04        13
          17       0.22      0.64      0.33       368
          18       0.03      0.15      0.04        27
          19       0.07      0.51      0.12        97
          20       0.29      0.65      0.40       551
          21       0.04      0.44      0.08        41
          22       0.05      0.46      0.09        85
          23       0.03      0.29      0.05        42
          24       0.06      0.57      0.12        83
          25       0.11      0.62      0.19       159
          26       0.15      0.68      0.24       112
          27       0.01      0.09      0.02        11
          28       0.28      0.59      0.38       596
          29       0.18      0.67      0.28       190
          30       0.11      0.66      0.19        83
          31       0.04      0.17      0.06        12
          32       0.06      0.38      0.10        26
          33       0.07      0.53      0.12        64
          34       0.04      0.36      0.07        25
          35       0.02      0.24      0.04        29
          36       0.11      0.74      0.20        92
          37       0.11      0.63      0.19       172
          38       0.10      0.57      0.17       136
          39       0.04      0.16      0.06        32
```

```
          40       0.05      0.35      0.08        40
          41       0.00      0.00      0.00         5
          42       0.07      0.59      0.13        93
          43       0.63      0.73      0.67      1155
          44       0.08      0.49      0.14       117
          45       0.16      0.75      0.27       145
          46       0.00      0.00      0.00         5
          47       0.13      0.66      0.22       129
          48       0.03      0.44      0.06        36
          49       0.03      0.41      0.06        44
          50       0.04      0.43      0.08        28
          51       0.06      0.57      0.11        51
          52       0.27      0.67      0.39       395
          53       0.05      0.42      0.09        65
          54       0.05      0.47      0.10        15
          55       0.02      0.27      0.04        37
          56       0.30      0.63      0.41       507
          57       0.38      0.65      0.48       587
          58       0.12      0.62      0.20       148
          59       0.14      0.63      0.23       173
          60       0.11      0.62      0.19        66
          61       0.04      0.41      0.08        51
          62       0.03      0.38      0.06        66
          63       0.02      0.31      0.04        39
          64       0.00      0.00      0.00         8
          65       0.16      0.65      0.26       228
          66       0.03      0.41      0.06        27
          67       0.06      0.48      0.11       119
          68       0.53      0.72      0.61       911
          69       0.10      0.29      0.14        14
          70       0.00      0.00      0.00        13

avg / total       0.27      0.61      0.35      9021

Time taken to run this cell : 2:03:44.677051

['71_tags_avg_w2v_lr.pkl']
```

Performance comparison of models

In [29]:

```
from prettytable import PrettyTable

print("Model performance with 71 tags (Hyperparamter tuning done in the previous notebook)")
print("="*83)
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 ', 0.0449,0.0628,0.3220,0.4212,0.3651])
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("\n\nModel performance with 71 tags, with the best model selected from the previous notebook + topic modell
ing")
print("="*105)
table =PrettyTable()
table.field_names = ["Model", "Vectorizer", "Accuracy", "Hamming loss","Precision","Recall","Micro F1"]
table.add_row(["LogisticRegression", 'TF-IDF 1-2 Grams + Topic Modelling ', 0.0455,0.0625,0.3231,0.4206,0.3684])
table.add_row(["LogisticRegression", 'Glove vectors + Topic Modelling ', 0.0003,0.1672,0.1471,0.6055,0.2367])
table.add_row(["LogisticRegression", 'Avg W2V + Topic Modelling ', 0.0003,0.1674,0.1483,0.6128,0.2388])
print(table)
```

```
Model performance with 71 tags (Hyperparamter tuning done in the previous notebook)
===================================================================================
+--------------------+-------------------+----------+--------------+-----------+--------+----------+
|       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 |  0.3651  |
| 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  |
+--------------------+-------------------+----------+--------------+-----------+--------+----------+


Model performance with 71 tags, with the best model selected from the previous notebook + topic mode
lling
=======================================================================================================
=====
+--------------------+-------------------------------------+----------+--------------+-----------+--
------+----------+
|       Model        |              Vectorizer             | Accuracy | Hamming loss | Precision | R
ecall | Micro F1 |
+--------------------+-------------------------------------+----------+--------------+-----------+--
------+----------+
| LogisticRegression | TF-IDF 1-2 Grams + Topic Modelling  |  0.0455  |    0.0625    |   0.3231  | 0
.4206 |  0.3684  |
| LogisticRegression |   Glove vectors + Topic Modelling   |  0.0003  |    0.1672    |   0.1471  | 0
.6055 |  0.2367  |
| LogisticRegression |      Avg W2V + Topic Modelling      |  0.0003  |    0.1674    |   0.1483  | 0
.6128 |  0.2388  |
+--------------------+-------------------------------------+----------+--------------+-----------+--
------+----------+
```

What we did in this experiment?

1. This is a continuation of the previous notebooks. We will use the pre-processed dataset that we have processed in the earlier notebook.
2. In this notebook we will use topic modelling along with the best model that we have obtained in the previous notebook to see if this approach increases the F1 score.
3. First we have extracted the top 8 topics from the corpus. We will some trial and error with 8, 15 and 22 topics and came to the conclusion that using 8 topics was giving us a better T-SNE representation of the dominant topics. Hence, we selected 8 topics and discarded the code for the rest of the trials.
4. We got our top 8 topics using LDA. LDA is nothing but a probabilistic approach to extract topics from a large set of corpus. Each corpus contains documents, each document contains topics.
5. Then we got the dominant topics keywords for each of the documents along with their topic coherence scores.
6. We then plotted a T-SNE plot to check how well these topics are segregated with respect to our data corpus.
7. Once we got the number of topics and a proper T-SNE visualization, we moved on to building our models with Logistic Regression. We have only used LR for 71 tags, because in the previous notebooks we have seen that LR performed better than all the other models we have tried.

Building the models.

1. In this case, we have vectorized the movie plots with TF-IDF N_Grams (1-2) and vectorized the topics using TF-IDF unigrams. We than used the scipy library to merge these two vectors and trained a model which would predict all the 71 tags. The F1-Score we got using this strategy gave us a minor improvement over the previous best model - 0.3684 vs 0.365. There's a very small improvement in the value of the micro averaged F1 score as compared to our previous best models for prediciting 71 tags.
2. In the second model that we built, we have vectorized the plot summaries using Glove vectors. We then combined the glove representations of each sentences to it's corresponding TF-IDF unigram representation of the topics. This approach resulted in a drop of the micro averaged F1 score value as compared to our best model - 0.2367 vs 0.3684. For obtaining the glove vectors, we have actually used a pre-trained model from Standford, since it's already trained on a very large corpus of data, it has better semantic representations.
3. In the third approach, we have used a pre-trained model from Google to obtain the average W2V representations of our movie summaries and combined the matrix with the TF-IDF unigram representation of the top topic keywords. This also didn't improve our previous best model and we could only get a micro averaged F1 score of 0.2388.

I had also experimented with bigram representation of the topic keywords but it reduced the micro averaged F1 score. Hence, I did not include the code for this as it was over-written by the bigram representation.

Conlusion:

Using topic modeling has improved our overall micro averaged F1 score just by a whisker - The previous best micro averaged F1 score was 0.3651. Using topic modeling and hyperparameter tuning, we got an overall F1 score of 0.3684, which is a bit satisfactory.

I expected the Glove vectors representation to perform better than the TF-IDF representations, since it takes into account the semantic meaning of words. But it didn't happen. On the contrary the F1 score decreased, I am not sure why this has happened.

I have used help and code from three resources mentioned below:

1. https://www.kaggle.com/konohayui/topic-modeling-on-quora-insincere-questions (https://www.kaggle.com/konohayui/topic-modeling-on-quora-insincere-questions)
2. https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/ (https://www.machinelearningplus.com/nlp/topic-modeling-gensim-python/)
3. https://www.youtube.com/watch?v=NYkbqzTlW3w (https://www.youtube.com/watch?v=NYkbqzTlW3w)

I have cited these references in the code sections where I have used them.

In [ ]:

```
import pdfkit
pdfkit.from_site('MPST Movie Plot Synopses Tag Prediction.html', 'MPST Movie Plot Synopses Tag Prediction.pdf')
```