In [1]:

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

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

In [14]:

```
#!pip install
```

In [15]:

```
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
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.feature_extraction.text import TfidfTransformer
from gensim import models
from gensim.models import Word2Vec
from gensim.models import KeyedVectors
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
from scipy.sparse import coo_matrix, hstack
from sklearn.model_selection import RandomizedSearchCV

# Gensim
import gensim
import gensim.corpora as corpora
from gensim.utils import simple_preprocess
from gensim.models import CoherenceModel
#keras
from keras.models import Sequential
from keras.layers import Dense,Dropout
from keras.layers import LSTM
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
from keras.preprocessing.text import Tokenizer
from keras.layers.normalization import BatchNormalization
from keras.layers import Input
from keras.layers import Flatten
from keras.preprocessing.sequence import pad_sequences
from keras.initializers import he_normal,glorot_normal
from keras.regularizers import l1,l2
from scipy.sparse import hstack
from keras.callbacks import ModelCheckpoint,EarlyStopping
from tensorboardcolab import *
from keras.optimizers import *
from keras.models import Model
from keras.layers import concatenate
from sklearn.preprocessing import LabelBinarizer, LabelEncoder
from keras.preprocessing import text, sequence
from keras import utils
```

In [20]:

```
pure_df = pd.read_csv('pure_df.csv')
pure_df.head(5)
```

Out[20]:

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
0	0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb	5	cult horror gothic murder atmospheric	tre volti dell paura
1	1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb	1	violence	dungeons dragons boo vile darknes
2	2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb	1	romantic	shop aroun corner
3	3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb	4	inspiring romantic stupid feel-good	mr holland opus
4	4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb	10	cruelty murder dramatic cult violence atmosphe...	scarface

Topic_Modelling

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

In [21]:

```
pure_df['pre_pro_plot_synopsis_words']=pure_df['pre_pro_plot_synopsis'].apply(lambda x: x.split())
pure_df.head()
```

Out[21]:

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
0	0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb	5	cult horror gothic murder atmospheric	tre volti dell paura
1	1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb	1	violence	dungeons dragons boo vile darknes
			The Shop	Matuschek's, a						

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
2	2	tt0033045	Around the Corner	gift store in Budapest, the ...	romantic	test	imdb	1	romantic	shop around corner
3	3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb	4	inspiring romantic stupid feel-good	mr holland opus
4	4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb	10	cruelty murder dramatic cult violence atmosphe...	scarface

In [0]:

```
# Create Dictionary
id2word = corpora.Dictionary(pure_df['pre_pro_plot_synopsis_words'])

# Create Corpus
texts = pure_df['pre_pro_plot_synopsis_words']

# Term Document Frequency
corpus = [id2word.doc2bow(text) for text in texts]
```

In [0]:

```
# View
print(corpus[:1])
```

```
[[(0, 1), (1, 2), (2, 1), (3, 1), (4, 1), (5, 1), (6, 2), (7, 2), (8, 1), (9, 1), (10, 2), (11, 1
), (12, 2), (13, 1), (14, 1), (15, 4), (16, 1), (17, 2), (18, 1), (19, 1), (20, 1), (21, 1), (22,
2), (23, 1), (24, 2), (25, 1), (26, 1), (27, 1), (28, 1), (29, 1), (30, 1), (31, 1), (32, 1), (33,
1), (34, 2), (35, 1), (36, 1), (37, 5), (38, 2), (39, 1), (40, 1), (41, 2), (42, 1), (43, 1), (44,
1), (45, 1), (46, 1), (47, 1), (48, 1), (49, 1), (50, 1), (51, 1), (52, 1), (53, 3), (54, 1), (55,
4), (56, 3), (57, 1), (58, 2), (59, 1), (60, 1), (61, 1), (62, 1), (63, 1), (64, 1), (65, 1), (66,
1), (67, 1), (68, 1), (69, 2), (70, 1), (71, 2), (72, 5), (73, 1), (74, 1), (75, 1), (76, 1), (77,
1), (78, 1), (79, 1), (80, 1), (81, 3), (82, 1), (83, 6), (84, 2), (85, 1), (86, 1), (87, 2), (88,
1), (89, 2), (90, 2), (91, 1), (92, 1), (93, 1), (94, 2), (95, 4), (96, 1), (97, 5), (98, 2), (99,
1), (100, 1), (101, 1), (102, 1), (103, 1), (104, 1), (105, 1), (106, 1), (107, 1), (108, 1), (109
, 1), (110, 2), (111, 1), (112, 1), (113, 1), (114, 1), (115, 1), (116, 1), (117, 1), (118, 2), (1
19, 1), (120, 1), (121, 1), (122, 1), (123, 1), (124, 1), (125, 1), (126, 1), (127, 1), (128, 1),
(129, 1), (130, 1), (131, 2), (132, 1), (133, 1), (134, 1), (135, 1), (136, 1), (137, 1), (138, 1)
, (139, 1), (140, 1), (141, 1), (142, 1), (143, 1), (144, 1), (145, 2), (146, 1), (147, 1), (148,
1), (149, 3), (150, 1), (151, 1), (152, 1), (153, 1), (154, 7), (155, 3), (156, 3), (157, 1), (158
, 2), (159, 1), (160, 1), (161, 2), (162, 3), (163, 1), (164, 1), (165, 1), (166, 1), (167, 1), (1
68, 1), (169, 1), (170, 3), (171, 1), (172, 1), (173, 1), (174, 1), (175, 1), (176, 5), (177, 1),
(178, 1), (179, 2), (180, 1), (181, 1), (182, 2), (183, 1), (184, 6), (185, 1), (186, 1), (187, 1)
, (188, 1), (189, 2), (190, 1), (191, 7), (192, 1), (193, 1), (194, 1), (195, 1), (196, 1), (197,
1), (198, 1), (199, 1), (200, 1), (201, 1), (202, 1), (203, 1), (204, 2), (205, 1), (206, 1), (207
, 4), (208, 1), (209, 2), (210, 1), (211, 1), (212, 1), (213, 1), (214, 1), (215, 1), (216, 1), (2
17, 1), (218, 1), (219, 1), (220, 2), (221, 2), (222, 1), (223, 1), (224, 1), (225, 1), (226, 3),
(227, 1), (228, 1), (229, 1), (230, 1), (231, 3), (232, 1), (233, 1), (234, 5), (235, 1), (236, 1)
, (237, 1), (238, 1), (239, 4), (240, 1), (241, 1), (242, 1), (243, 1), (244, 1), (245, 1), (246,
2), (247, 1), (248, 1), (249, 1), (250, 1), (251, 2), (252, 1), (253, 1), (254, 1), (255, 2), (256
, 1), (257, 1), (258, 1), (259, 2), (260, 1), (261, 1), (262, 1), (263, 4), (264, 1), (265, 2), (2
66, 1), (267, 1), (268, 6), (269, 1), (270, 1), (271, 1), (272, 1), (273, 2), (274, 1), (275, 1),
(276, 1), (277, 1), (278, 1), (279, 1), (280, 1), (281, 1), (282, 1), (283, 1), (284, 1), (285, 1)
, (286, 1), (287, 1), (288, 7), (289, 4), (290, 2), (291, 1), (292, 2), (293, 1), (294, 1), (295,
2), (296, 6), (297, 1), (298, 1), (299, 1), (300, 1), (301, 3), (302, 5), (303, 1), (304, 1), (305
, 1), (306, 1), (307, 1), (308, 1), (309, 1), (310, 1), (311, 1), (312, 1), (313, 2), (314, 1), (3
15, 1), (316, 4), (317, 1), (318, 2), (319, 1), (320, 1), (321, 1), (322, 1), (323, 1), (324, 1),
(325, 1), (326, 1), (327, 1), (328, 1), (329, 1), (330, 1), (331, 2), (332, 1), (333, 1), (334, 1)
, (335, 1), (336, 1), (337, 1), (338, 1), (339, 1), (340, 2), (341, 1), (342, 1), (343, 1), (344,
1), (345, 1), (346, 1), (347, 1), (348, 1), (349, 1), (350, 1), (351, 1), (352, 4), (353, 1), (354
, 1), (355, 1), (356, 2), (357, 1), (358, 1), (359, 3), (360, 1), (361, 1), (362, 13), (363, 1), (
364, 1), (365, 1), (366, 1), (367, 1), (368, 1), (369, 1), (370, 3), (371, 1), (372, 7), (373, 1),
(374, 1), (375, 1), (376, 1), (377, 1), (378, 1), (379, 1), (380, 1), (381, 1), (382, 1), (383, 1)
, (384, 1), (385, 1), (386, 1), (387, 1), (388, 1), (389, 2), (390, 1), (391, 3), (392, 1), (393,
1), (394, 2), (395, 2), (396, 2), (397, 1), (398, 1), (399, 1), (400, 1), (401, 1), (402, 1), (403
, 1), (404, 1), (405, 1), (406, 2), (407, 1), (408, 1), (409, 1), (410, 1), (411, 3), (412, 1), (4
```

```
13, 2), (414, 1), (415, 1), (416, 1), (417, 1), (418, 2), (419, 1), (420, 1), (421, 1), (422, 1),
(423, 1), (424, 1), (425, 1), (426, 1), (427, 1), (428, 2), (429, 1), (430, 1), (431, 2), (432, 1)
, (433, 1), (434, 1), (435, 1), (436, 2), (437, 1), (438, 1), (439, 3), (440, 1), (441, 1), (442,
2), (443, 1), (444, 1), (445, 1), (446, 1), (447, 1), (448, 1), (449, 1), (450, 3), (451, 1), (452
, 1), (453, 1), (454, 1), (455, 1), (456, 1), (457, 1), (458, 1), (459, 1), (460, 1), (461, 1), (4
62, 1), (463, 10), (464, 2), (465, 1), (466, 1), (467, 2), (468, 1), (469, 2), (470, 1), (471, 1),
(472, 2), (473, 1), (474, 1), (475, 3), (476, 1), (477, 1), (478, 1), (479, 1), (480, 1), (481, 1)
, (482, 4), (483, 1), (484, 4), (485, 1)]]
```

In [0]:

```
# Human readable format of corpus (term-frequency)
[[(id2word[id], freq) for id, freq in cp] for cp in corpus[:1]]
```

Out[0]:

```
[[('19th', 1),
  ('abandoned', 2),
  ('abby', 1),
  ('absence', 1),
  ('acquisition', 1),
  ('actually', 1),
  ('agrees', 2),
  ('aid', 2),
  ('air', 1),
  ('alfonsi', 1),
  ('ali', 2),
  ('already', 1),
  ('also', 2),
  ('anderson', 1),
  ('anticipating', 1),
  ('apartment', 4),
  ('apparently', 1),
  ('appearance', 2),
  ('appears', 1),
  ('arms', 1),
  ('around', 1),
  ('arrival', 1),
  ('arrives', 2),
  ('ask', 1),
  ('assailed', 2),
  ('attacked', 1),
  ('attempts', 1),
  ('attracted', 1),
  ('attractive', 1),
  ('await', 1),
  ('awaken', 1),
  ('awakened', 1),
  ('awakens', 1),
  ('away', 1),
  ('back', 2),
  ('basement', 1),
  ('battle', 1),
  ('bed', 5),
  ('beg', 2),
  ('begin', 1),
  ('beginning', 1),
  ('begs', 2),
  ('beheaded', 1),
  ('beheads', 1),
  ('behind', 1),
  ('believes', 1),
  ('belongs', 1),
  ('beset', 1),
  ('best', 1),
  ('bite', 1),
  ('bites', 1),
  ('bits', 1),
  ('blade', 1),
  ('blood', 3),
  ('bode', 1),
  ('body', 4),
  ('boris', 3),
  ('branches', 1),
  ('breaks', 2),
  ('breakup', 1),
```

```
('brother', 1),
('brothers', 1),
('bruise', 1),
('burial', 1),
('bury', 1),
('busy', 1),
('butcher', 1),
('buzzing', 1),
('cadaver', 1),
('call', 2),
('called', 1),
('caller', 2),
('calls', 5),
('calm', 1),
('camera', 1),
('careful', 1),
('caring', 1),
('case', 1),
('cathedral', 1),
('caused', 1),
('century', 1),
('certain', 3),
('change', 1),
('chester', 6),
('child', 2),
('close', 1),
('come', 1),
('coming', 2),
('commit', 1),
('concierge', 2),
('concludes', 2),
('confession', 1),
('confirmed', 1),
('confused', 1),
('continues', 2),
('corpse', 4),
('corpses', 1),
('cottage', 5),
('count', 2),
('course', 1),
('creeps', 1),
('crew', 1),
('crewmen', 1),
('curse', 1),
('daggers', 1),
('daughter', 1),
('dawn', 1),
('days', 1),
('de', 1),
('dead', 1),
('death', 2),
('decides', 1),
('decision', 1),
('demeanor', 1),
('dialina', 1),
('diamond', 1),
('died', 1),
('difficult', 1),
('discovered', 2),
('discovers', 1),
('distinct', 1),
('distracted', 1),
('distressed', 1),
('doctor', 1),
('door', 1),
('doubt', 1),
('drains', 1),
('dreaded', 1),
('dressed', 1),
('dripping', 1),
('drop', 1),
('drops', 2),
('duty', 1),
('eagerly', 1),
('earlier', 1),
('east', 1),
('elaborate', 1),
```

```
    ('elderly', 1),
    ('embraces', 1),
    ('end', 1),
    ('england', 1),
    ('entrance', 1),
    ('escape', 1),
    ('escaped', 1),
    ('estranged', 1),
    ('evening', 2),
    ('events', 1),
    ('ex', 1),
    ('examine', 1),
    ('explains', 3),
    ('face', 1),
    ('faces', 1),
    ('fails', 1),
    ('fake', 1),
    ('family', 7),
    ('father', 3),
    ('fear', 3),
    ('fears', 1),
    ('feeds', 2),
    ('felt', 1),
    ('final', 1),
    ('finds', 2),
    ('finger', 3),
    ('finishes', 1),
    ('fit', 1),
    ('five', 1),
    ('flat', 1),
    ('flee', 1),
    ('flees', 1),
    ('floor', 1),
    ('fly', 3),
    ('follow', 1),
    ('forcing', 1),
    ('forest', 1),
    ('forgiveness', 1),
    ('former', 1),
    ('frank', 5),
    ('franks', 1),
    ('friends', 1),
    ('fright', 2),
    ('front', 1),
    ('gasps', 1),
    ('gets', 2),
    ('ghosts', 1),
    ('giorgio', 6),
    ('girl', 1),
    ('gives', 1),
    ('glass', 1),
    ('glauco', 1),
    ('go', 2),
    ('gone', 1),
    ('gorcha', 7),
    ('grave', 1),
    ('greed', 1),
    ('greeted', 1),
    ('gripping', 1),
    ('gustavo', 1),
    ('hands', 1),
    ('hanging', 1),
    ('happens', 1),
    ('happy', 1),
    ('harriet', 1),
    ('heart', 1),
    ('helen', 1),
    ('help', 2),
    ('hide', 1),
    ('high', 1),
    ('home', 4),
    ('horror', 1),
    ('horse', 2),
    ('house', 1),
    ('however', 1),
    ('husband', 1),
    ('hysteria', 1),
```

```
 ('identified', 1),
 ('image', 1),
 ('imaging', 1),
 ('immediately', 1),
 ('impersonating', 1),
 ('infected', 1),
 ('infects', 2),
 ('introduces', 2),
 ('intruder', 1),
 ('investigator', 1),
 ('invited', 1),
 ('italian', 1),
 ('ivan', 3),
 ('jacqueline', 1),
 ('jail', 1),
 ('job', 1),
 ('journey', 1),
 ('karloff', 3),
 ('kill', 1),
 ('killed', 1),
 ('knife', 5),
 ('knowing', 1),
 ('known', 1),
 ('landed', 1),
 ('large', 1),
 ('later', 4),
 ('law', 1),
 ('learned', 1),
 ('leave', 1),
 ('left', 1),
 ('lesbian', 1),
 ('lies', 1),
 ('life', 2),
 ('lights', 1),
 ('likely', 1),
 ('living', 1),
 ('london', 1),
 ('long', 2),
 ('longer', 1),
 ('looking', 1),
 ('love', 1),
 ('lover', 2),
 ('lovers', 1),
 ('loving', 1),
 ('lured', 1),
 ('lying', 2),
 ('lynda', 1),
 ('macabre', 1),
 ('maddening', 1),
 ('makes', 4),
 ('making', 1),
 ('man', 2),
 ('mans', 1),
 ('marry', 1),
 ('mary', 6),
 ('massimo', 1),
 ('matter', 1),
 ('meant', 1),
 ('medin', 1),
 ('medium', 2),
 ('members', 1),
 ('mercier', 1),
 ('michele', 1),
 ('midnight', 1),
 ('morning', 1),
 ('mostly', 1),
 ('mother', 1),
 ('motionless', 1),
 ('moving', 1),
 ('murderous', 1),
 ('nardo', 1),
 ('necessary', 1),
 ('nerves', 1),
 ('next', 1),
 ('night', 7),
 ('no', 4),
 ('nobleman', 2),
```

```
('not', 1),
('note', 2),
('notes', 1),
('notice', 1),
('notices', 2),
('nurse', 6),
('nylon', 1),
('observation', 1),
('odor', 1),
('offers', 1),
('old', 3),
('one', 5),
('onorato', 1),
('optimistic', 1),
('order', 1),
('orginal', 1),
('outlaw', 1),
('panic', 1),
('parisian', 1),
('pathologist', 1),
('pens', 1),
('pester', 1),
('phone', 2),
('phones', 1),
('pierreux', 1),
('pietro', 4),
('pillow', 1),
('pimp', 2),
('placed', 1),
('pleased', 1),
('plunged', 1),
('police', 1),
('possibility', 1),
('precautions', 1),
('preferably', 1),
('prepare', 1),
('prevent', 1),
('prevented', 1),
('priced', 1),
('pried', 1),
('prison', 2),
('promising', 1),
('prop', 1),
('protection', 1),
('pulls', 1),
('puts', 1),
('quickly', 1),
('reach', 1),
('real', 1),
('realize', 2),
('realizes', 1),
('reason', 1),
('recently', 1),
('regularity', 1),
('release', 1),
('relinquish', 1),
('reluctant', 1),
('reluntantly', 1),
('rest', 1),
('return', 1),
('returning', 1),
('returns', 4),
('reveal', 1),
('revenge', 1),
('reviving', 1),
('riding', 2),
('righi', 1),
('rika', 1),
('ring', 3),
('rises', 1),
('room', 1),
('rosy', 13),
('rosys', 1),
('ruins', 1),
('run', 1),
('runs', 1),
('rural', 1),
```

```
 ('rushes', 1),
 ('russia', 1),
 ('scene', 3),
 ('screams', 1),
 ('sdenka', 7),
 ('seconds', 1),
 ('seen', 1),
 ('sees', 1),
 ('segment', 1),
 ('segments', 1),
 ('seizes', 1),
 ('series', 1),
 ('several', 1),
 ('shelter', 1),
 ('show', 1),
 ('siblings', 1),
 ('sign', 1),
 ('simple', 1),
 ('simulate', 1),
 ('sister', 1),
 ('sitting', 1),
 ('sleeps', 2),
 ('slowly', 1),
 ('small', 3),
 ('solace', 1),
 ('someone', 1),
 ('son', 2),
 ('soon', 2),
 ('sound', 2),
 ('sounds', 1),
 ('sour', 1),
 ('souvenir', 1),
 ('space', 1),
 ('spacious', 1),
 ('splash', 1),
 ('stabbing', 1),
 ('stabs', 1),
 ('stakes', 1),
 ('stay', 2),
 ('steals', 1),
 ('stockings', 1),
 ('stop', 1),
 ('stops', 1),
 ('strange', 3),
 ('strangle', 1),
 ('strangles', 2),
 ('stroke', 1),
 ('strong', 1),
 ('struck', 1),
 ('struggle', 1),
 ('subsequently', 2),
 ('suggestion', 1),
 ('suicide', 1),
 ('suite', 1),
 ('supernatural', 1),
 ('surprised', 1),
 ('surrounded', 1),
 ('susy', 1),
 ('swooping', 1),
 ('synopsis', 1),
 ('taken', 2),
 ('takes', 1),
 ('taking', 1),
 ('tales', 2),
 ('telephonerosy', 1),
 ('tells', 1),
 ('tempted', 1),
 ('term', 1),
 ('terrified', 2),
 ('testimony', 1),
 ('threatens', 1),
 ('three', 3),
 ('throat', 1),
 ('ties', 1),
 ('time', 2),
 ('tips', 1),
 ('took', 1),
```

```
  ('torn', 1),
  ('towards', 1),
  ('tranquillizer', 1),
  ('transpires', 1),
  ('trip', 1),
  ('two', 3),
  ('ultimately', 1),
  ('unkempt', 1),
  ('unknown', 1),
  ('unsettled', 1),
  ('urfe', 1),
  ('vacant', 1),
  ('vampires', 1),
  ('various', 1),
  ('victorian', 1),
  ('viewers', 1),
  ('violence', 1),
  ('visible', 1),
  ('vladimir', 10),
  ('walking', 2),
  ('walls', 1),
  ('warn', 1),
  ('water', 2),
  ('waterin', 1),
  ('way', 2),
  ('well', 1),
  ('white', 1),
  ('wife', 2),
  ('withdraws', 1),
  ('without', 1),
  ('woman', 3),
  ('womans', 1),
  ('women', 1),
  ('worse', 1),
  ('would', 1),
  ('writing', 1),
  ('wrong', 1),
  ('wurdalak', 4),
  ('wurdalakin', 1),
  ('young', 4),
  ('younger', 1)]]
```

In [0]:

```
#We need to run below steps before giving path to mallet because colab cannot access mallet from d
rive so we
#need to upload entire mallet files here and also upgrade the gensim and install java for mallet

#----------------------------step-1----------------------------------------------------- -
'''
!pip install --upgrade gensim


#-----------------------------step-2---------------------------------------------------------

import os        #importing os to set environment variable
def install_java():
  !apt-get install -y openjdk-8-jdk-headless -qq > /dev/null      #install openjdk
  os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"     #set environment variable
  !java -version       #check java version
install_java()


#-----------------------------step-3----------------------------------------------------------
#below command directly download mallet zip file  into colab disk next line is we can unzip it and
use that path to mallet

!wget http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip
!unzip mallet-2.0.8.zip
'''
#refer this https://github.com/polsci/colab-gensim-mallet
```

Out[0]:

```
'\n!pip install --upgrade gensim\n\n\n#-----------------------------step-2--------------------------
-----------------------------\n\nimport os        #importing os to set environment variable\ndef ins
```

```
-----------------------------\n\nimport os        #importing os to set environment variable\ndef ins
all_java():\n  !apt-get install -y openjdk-8-jdk-headless -qq > /dev/null      #install openjdk\n
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"     #set environment variable\n
!java -version       #check java version\ninstall_java()\n\n\n#---------------------------step-3--
--------------------------------------------------------\n#below command directly download mallet zi
file  into colab disk next line is we can unzip it and use that path to mallet \n\n!wget
http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip\n!unzip mallet-2.0.8.zip\n'
```

In [0]:

```python
# Download File: http://mallet.cs.umass.edu/dist/mallet-2.0.8.zip
mallet_path = '/content/mallet-2.0.8/bin/mallet' # update this path
```

In [0]:

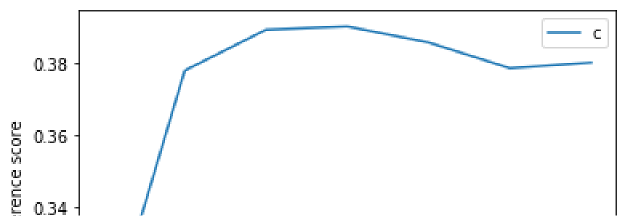
```python
def compute_coherence_values(dictionary, corpus, texts, limit, start=2, step=3):
    """
    Compute c_v coherence for various number of topics

    Parameters:
    ----------
    dictionary : Gensim dictionary
    corpus : Gensim corpus
    texts : List of input texts
    limit : Max num of topics

    Returns:
    -------
    model_list : List of LDA topic models
    coherence_values : Coherence values corresponding to the LDA model with respective number of t
opics
    """
    coherence_values = []
    model_list = []
    for num_topics in range(start, limit, step):
        model = gensim.models.wrappers.LdaMallet(mallet_path, corpus=corpus, num_topics=num_topics,
id2word=id2word)
        model_list.append(model)
        coherencemodel = CoherenceModel(model=model, texts=texts, dictionary=dictionary, coherence=
'c_v')
        coherence_values.append(coherencemodel.get_coherence())

    return model_list, coherence_values
```

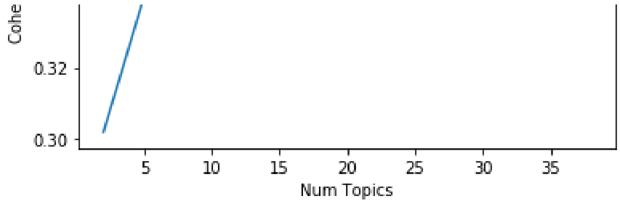
In [0]:

```python
# Can take a long time to run.
model_list, coherence_values = compute_coherence_values(dictionary=id2word, corpus=corpus,
                                                        texts=pure_df['pre_pro_plot_synopsis_words'
                                                        start=2, limit=40, step=6)
```

In [0]:

```python
# Show graph
limit=40; start=2; step=6;
x = range(start, limit, step)
plt.plot(x, coherence_values)
plt.xlabel("Num Topics")
plt.ylabel("Coherence score")
plt.legend(("coherence_values"), loc='best')
plt.show()
# Print the coherence scores
for m, cv in zip(x, coherence_values):
    print("Num Topics =", m, " has Coherence Value of", round(cv, 4))
```






```
Num Topics = 2  has Coherence Value of 0.302
Num Topics = 8  has Coherence Value of 0.378
Num Topics = 14  has Coherence Value of 0.3894
Num Topics = 20  has Coherence Value of 0.3903
Num Topics = 26  has Coherence Value of 0.3858
Num Topics = 32  has Coherence Value of 0.3786
Num Topics = 38  has Coherence Value of 0.3802
```

In [0]:

```
# Build LDA model
lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus,id2word=id2word,num_topics=20,random_stat
e=100,update_every=1,chunksize=100,passes=10,alpha='auto',
                                            per_word_topics=True)
```

In [0]:

```
# Print the Keyword in the 10 topics
print(lda_model.print_topics())
doc_lda = lda_model[corpus]
data = pure_df['pre_pro_plot_synopsis'].tolist()
```

```
[(0, '0.009*"earth" + 0.009*"ship" + 0.008*"world" + 0.006*"island" + 0.006*"crew" + 0.005*"betty"
+ 0.005*"dr" + 0.005*"control" + 0.005*"city" + 0.005*"human"'), (1, '0.007*"king" + 0.006*"fight"
+ 0.005*"one" + 0.005*"death" + 0.005*"kill" + 0.004*"however" + 0.004*"help" + 0.003*"two" + 0.00
3*"queen" + 0.003*"battle"'), (2, '0.066*"charlie" + 0.044*"joe" + 0.038*"frank" + 0.029*"helen" +
0.021*"jeff" + 0.020*"kit" + 0.017*"emma" + 0.017*"c" + 0.015*"n" + 0.015*"nina"'), (3,
'0.012*"war" + 0.007*"men" + 0.006*"army" + 0.006*"soldiers" + 0.006*"american" + 0.005*"british"
+ 0.005*"captain" + 0.005*"group" + 0.005*"german" + 0.005*"general"'), (4, '0.025*"eric" +
0.022*"matt" + 0.021*"casey" + 0.021*"arthur" + 0.018*"jenny" + 0.015*"zombies" + 0.014*"ogami" +
0.013*"godfrey" + 0.012*"wolf" + 0.011*"vera"'), (5, '0.045*"george" + 0.034*"jerry" +
0.025*"chris" + 0.024*"bugs" + 0.017*"roy" + 0.016*"kate" + 0.015*"anna" + 0.014*"logan" +
0.011*"joanna" + 0.011*"jonathan"'), (6, '0.022*"max" + 0.021*"johnny" + 0.016*"adam" +
0.014*"ghost" + 0.013*"daniel" + 0.011*"witch" + 0.010*"white" + 0.009*"devil" + 0.008*"god" + 0.0
08*"monster"'), (7, '0.015*"prison" + 0.010*"nick" + 0.010*"bernard" + 0.009*"charles" +
0.009*"dant" + 0.008*"wilson" + 0.007*"count" + 0.007*"philip" + 0.006*"judge" +
0.006*"sherman"'), (8, '0.034*"john" + 0.018*"david" + 0.012*"doctor" + 0.012*"richard" +
0.010*"sarah" + 0.010*"robert" + 0.009*"dr" + 0.009*"ann" + 0.007*"elizabeth" + 0.006*"william"'),
(9, '0.038*"mr" + 0.035*"henry" + 0.024*"sir" + 0.023*"mrs" + 0.023*"rachel" + 0.022*"jane" + 0.01
6*"brown" + 0.015*"barbara" + 0.015*"batman" + 0.014*"bruce"'), (10, '0.018*"police" +
0.011*"jack" + 0.009*"money" + 0.008*"kill" + 0.008*"killed" + 0.007*"car" + 0.007*"sam" + 0.007*"
gang" + 0.006*"man" + 0.005*"murder"'), (11, '0.010*"valjean" + 0.007*"goku" + 0.007*"son" + 0.006
*"om" + 0.006*"kishen" + 0.006*"jin" + 0.005*"th" + 0.005*"gauri" + 0.005*"ajay" +
0.005*"india"'), (12, '0.014*"house" + 0.008*"find" + 0.008*"back" + 0.008*"finds" + 0.007*"body"
+ 0.007*"car" + 0.007*"room" + 0.007*"man" + 0.005*"night" + 0.005*"dead"'), (13, '0.042*"michael"
+ 0.024*"tommy" + 0.019*"steve" + 0.017*"todd" + 0.016*"tony" + 0.013*"leo" + 0.012*"anne" +
0.011*"albert" + 0.009*"el" + 0.009*"frankie"'), (14, '0.031*"not" + 0.023*"tells" + 0.014*"back"
+ 0.012*"get" + 0.011*"asks" + 0.011*"go" + 0.010*"tom" + 0.010*"goes" + 0.010*"says" +
0.008*"gets"'), (15, '0.012*"school" + 0.008*"new" + 0.006*"bill" + 0.006*"show" + 0.005*"game" +
0.005*"jimmy" + 0.004*"friends" + 0.004*"fred" + 0.004*"high" + 0.004*"jason"'), (16,
'0.028*"town" + 0.013*"men" + 0.012*"ben" + 0.011*"billy" + 0.010*"sheriff" + 0.008*"jean" + 0.008
*"horse" + 0.008*"gold" + 0.007*"hamlet" + 0.007*"joseph"'), (17, '0.029*"paul" + 0.024*"harry" +
0.024*"alice" + 0.019*"alex" + 0.019*"lucy" + 0.015*"maria" + 0.014*"amy" + 0.013*"edward" + 0.011
*"marie" + 0.010*"arisen"'), (18, '0.028*"jim" + 0.028*"peter" + 0.016*"scott" + 0.015*"red" + 0.0
10*"cat" + 0.010*"spider" + 0.009*"gus" + 0.009*"bart" + 0.008*"silver" + 0.007*"porky"'), (19, '0
.011*"father" + 0.010*"not" + 0.008*"family" + 0.008*"mother" + 0.008*"love" + 0.007*"one" + 0.007
*"life" + 0.006*"home" + 0.006*"time" + 0.006*"film"')]
```

In [0]:

```
%%time
def format_topics_sentences(ldamodel=lda_model, 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
        row = sorted(row, key=lambda x: (x[1]), reverse=True)
        # Get the Dominant topic, Perc Contribution and Keywords for each document
        for j, (topic_num, prop_topic) in enumerate(row):
            if j == 0:  # => dominant topic
                wp = ldamodel.show_topic(topic_num)
                topic_keywords = ", ".join([word for word, prop in wp])
                sent_topics_df = sent_topics_df.append(pd.Series([int(topic_num),
round(prop_topic,4), topic_keywords]), ignore_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)

# Format
df_dominant_topic = df_topic_sents_keywords.reset_index()
df_dominant_topic.columns = ['Document_No', 'Dominant_Topic', 'Topic_Perc_Contrib', 'Keywords', 'T
ext']

# Show
df_dominant_topic.head(10)
```

```
CPU times: user 9min 59s, sys: 3min 19s, total: 13min 19s
Wall time: 9min 31s
```

In [0]:

```
df_dominant_topic.to_csv('drive/My Drive/colab/topic_modelling_data.csv',index=False)
```

In [22]:

```
df_dominant_topic =  pd.read_csv('topic_modelling_data.csv')
```

In [23]:

```
pure_df.head(3)
```

Out[23]:

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
0	0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb	5	cult horror gothic murder atmospheric	tre volti dell paura
1	1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb	1	violence	dungeons dragons boo vile darknes
2	2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb	1	romantic	shop aroun corner

In [24]:

```
df_dominant_topic.head(3)
```

Out[24]:

	Document_No	Dominant_Topic	Topic_Perc_Contrib	Keywords	Text
0	0	12.0	0.2685	house, find, back, finds, body, car, room, man...	note synopsis orginal italian release segments...
1	1	1.0	0.3064	king, fight, one, death, kill, however, help, ...	two thousand years ago nhagruul foul sorcerer ...
2	2	19.0	0.3600	father, not, family, mother, love, one, life, ...	matuschek gift store budapest workplace alfred...

In [25]:

```
final_model=pd.concat([pure_df,df_dominant_topic], axis=1)
final_model.head()
```

Out[25]:

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
0	0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb	5	cult horror gothic murder atmospheric	tre volti dell paura
1	1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb	1	violence	dungeons dragons boo vile darknes
2	2	tt0033045	The Shop Around the Corner	Matuschek's, a gift store in Budapest, is the ...	romantic	test	imdb	1	romantic	shop aroun corner
3	3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb	4	inspiring romantic stupid feel-good	mr holland opus
4	4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb	10	cruelty murder dramatic cult violence atmosphe...	scarface

In [0]:

```
x_train = final_model.loc[(final_model['split'] == 'train')]
x_cv = final_model.loc[(final_model['split'] == 'val')]
x_test = final_model.loc[(final_model['split'] == 'test')]

y_train_71 = x_train['pre_pro_tags']
y_cv_71 = x_cv['pre_pro_tags']
y_test_71 = x_test['pre_pro_tags']
```

```
#Convert the tags to binary vectors
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(" "), binary='true')
y_train_71 = vectorizer.fit_transform(y_train_71)
y_cv_71 = vectorizer.transform(y_cv_71)
y_test_71 = vectorizer.transform(y_test_71)

print('train_data shape',x_train.shape,y_train_71.shape)
print('train_data shape',x_cv.shape,y_cv_71.shape)
print('test_data shape',x_test.shape,y_test_71.shape)
```

```
train_data shape (9435, 17) (9435, 71)
train_data shape (2362, 17) (2362, 71)
test_data shape (2955, 17) (2955, 71)
```

Bow

In [0]:

```
#plots + topics
#plots
%%time
vectorizer = CountVectorizer(min_df=10)
xb_train_multilabel = vectorizer.fit_transform(x_train['pre_pro_plot_synopsis'])
xb_cv_multilabel = vectorizer.transform(x_cv['pre_pro_plot_synopsis'])
xb_test_multilabel = vectorizer.transform(x_test['pre_pro_plot_synopsis'])
```

```
CPU times: user 5.9 s, sys: 19 ms, total: 5.92 s
Wall time: 5.93 s
```

In [0]:

```
#topics
%%time
vectorizer = CountVectorizer(min_df=10)
xk_train_multilabel = vectorizer.fit_transform(x_train['Keywords'])
xk_cv_multilabel = vectorizer.transform(x_cv['Keywords'])
xk_test_multilabel = vectorizer.transform(x_test['Keywords'])
```

```
CPU times: user 170 ms, sys: 0 ns, total: 170 ms
Wall time: 175 ms
```

In [0]:

```
#topics + plots
from scipy.sparse import hstack
xb_topic_plot_train=hstack([xb_train_multilabel,xk_train_multilabel])
xb_topic_plot_cv=hstack([xb_cv_multilabel,xk_cv_multilabel])
xb_topic_plot_test=hstack([xb_test_multilabel,xk_test_multilabel])

print('bow_train data',xb_topic_plot_train.shape,y_train_71.shape)
print('bow_cv data',xb_topic_plot_cv.shape,y_cv_71.shape)
print('bow_test data',xb_topic_plot_test.shape,y_test_71.shape)
```

```
bow_train data (9435, 21404) (9435, 71)
bow_cv data (2362, 21404) (2362, 71)
bow_test data (2955, 21404) (2955, 71)
```

In [0]:

```
#hyperparameter tuning
#we have multiple models to train so we create a model function
def log_reg(x_train,y_train,x_cv,y_cv,x_test,y_test):
  train_f1 = []
  cv_f1 = []
  parameters=[0.0001,0.001,0.01,0.1,1,10,100,1000]
  for i in parameters:
    classifier = OneVsRestClassifier(LogisticRegression(C=i, penalty='l1',class_weight='balanced'))
    classifier.fit(x_train, y_train)
```

```
    train_predictions = classifier.predict (x_train)
    train_f1_score = f1_score(y_train, train_predictions, average='micro')
    train_f1.append(train_f1_score)
    cv_predictions = classifier.predict(x_cv)
    cv_f1_score = f1_score(y_cv, cv_predictions, average='micro')
    cv_f1.append(cv_f1_score)
    print("for",i,      "Train_f1_score: {:.4f}, Cv_f1_score: {:.4f}".format(train_f1_score,
cv_f1_score))
  best_estimators = np.argmax(cv_f1)
  print('best parameter :',parameters[best_estimators])
  #modeling with test data with best hyper paremeter
  classifier2 = OneVsRestClassifier(LogisticRegression(C=parameters[best_estimators], penalty='l1',
class_weight='balanced'))
  classifier2.fit(x_train, y_train)
  predictions = classifier2.predict(x_test)

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

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

  print("Micro-average :")
  print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

In [0]:

```
log_reg(xb_topic_plot_train,y_train_71,xb_topic_plot_cv,y_cv_71,xb_topic_plot_test,y_test_71)
```

```
for 0.0001 Train_f1_score: 0.0061, Cv_f1_score: 0.0072
for 0.001 Train_f1_score: 0.1159, Cv_f1_score: 0.1172
for 0.01 Train_f1_score: 0.3814, Cv_f1_score: 0.2866
for 0.1 Train_f1_score: 0.7747, Cv_f1_score: 0.3188
for 1 Train_f1_score: 0.9723, Cv_f1_score: 0.2987
for 10 Train_f1_score: 0.9731, Cv_f1_score: 0.2906
for 100 Train_f1_score: 0.9731, Cv_f1_score: 0.2896
for 1000 Train_f1_score: 0.9731, Cv_f1_score: 0.2825
best parameter : 0.1
Accuracy : 0.028764805414551606
Hamming loss  0.06989347251018803
Micro-average :
Precision: 0.2763, Recall: 0.3878, F1-measure: 0.3227
```

TF IDF

In [0]:

```
#plots + topics
#plots
%%time
vectorizer = TfidfVectorizer(min_df=10)
xt_train_multilabel = vectorizer.fit_transform(x_train['pre_pro_plot_synopsis'])
xt_cv_multilabel = vectorizer.transform(x_cv['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer.transform(x_test['pre_pro_plot_synopsis'])

#topics
vectorizer = TfidfVectorizer(min_df=10)
xtk_train_multilabel = vectorizer.fit_transform(x_train['Keywords'])
xtk_cv_multilabel = vectorizer.transform(x_cv['Keywords'])
xtk_test_multilabel = vectorizer.transform(x_test['Keywords'])

#topics + plots
xt_topic_plot_train=hstack([xt_train_multilabel,xtk_train_multilabel])
xt_topic_plot_cv=hstack([xt_cv_multilabel,xtk_cv_multilabel])
xt_topic_plot_test=hstack([xt_test_multilabel,xtk_test_multilabel])
```

```
CPU times: user 6.39 s, sys: 22.1 ms, total: 6.41 s
Wall time: 6.42 s
```

In [0]:

```
log_reg(xt_topic_plot_train,y_train_71,xt_topic_plot_cv,y_cv_71,xt_topic_plot_test,y_test_71)
```

```
for 0.0001 Train_f1_score: 0.0000, Cv_f1_score: 0.0000
for 0.001 Train_f1_score: 0.0000, Cv_f1_score: 0.0000
for 0.01 Train_f1_score: 0.1168, Cv_f1_score: 0.1240
for 0.1 Train_f1_score: 0.1956, Cv_f1_score: 0.1817
for 1 Train_f1_score: 0.5446, Cv_f1_score: 0.3103
for 10 Train_f1_score: 0.9355, Cv_f1_score: 0.3012
for 100 Train_f1_score: 0.9730, Cv_f1_score: 0.2911
for 1000 Train_f1_score: 0.9731, Cv_f1_score: 0.2885
best parameter : 1
Accuracy : 0.013874788494077835
Hamming loss  0.08743356926670003
Micro-average :
Precision: 0.2349, Recall: 0.4591, F1-measure: 0.3108
```

Tuning Parameters with GridSearch using TFIDF

In [38]:

```
train_data  = final_model.loc[(final_model['split'] == 'train') | (final_model['split'] == 'val')]
test_data = final_model.loc[(final_model['split'] == 'test')]
y_train = train_data['pre_pro_tags']
y_test = test_data['pre_pro_tags']
```

In [27]:

```
train_data.head(3)
```

Out[27]:

	Unnamed: 0	imdb_id	title	plot_synopsis	tags	split	synopsis_source	tags_count	tags_2	pre_pro_tit
0	0	tt0057603	I tre volti della paura	Note: this synopsis is for the orginal Italian...	cult, horror, gothic, murder, atmospheric	train	imdb	5	cult horror gothic murder atmospheric	tre volti dell paura
1	1	tt1733125	Dungeons & Dragons: The Book of Vile Darkness	Two thousand years ago, Nhagruul the Foul, a s...	violence	train	imdb	1	violence	dungeons dragons boo vile darknes
3	3	tt0113862	Mr. Holland's Opus	Glenn Holland, not a morning person by anyone'...	inspiring, romantic, stupid, feel-good	train	imdb	4	inspiring romantic stupid feel-good	mr holland opus

In [19]:

```
#Computing Grid/Random search is computational expensive on BOW Vectorizer so we trying on TFIDF a
ccoring to research paper
#TFIDF UNI GRAMS
#plots + keywords
#keywords

vectorizer_1 = 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))
xt_train_multilabel = vectorizer_1.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_1.transform(test_data['pre_pro_plot_synopsis'])
```

```
#topics
vectorizer_2 = TfidfVectorizer(smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), subl
inear_tf=False,
                             ngram_range=(1,1))
xtk_train_multilabel = vectorizer_2.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_2.transform(test_data['Keywords'])

vectorizer = CountVectorizer(tokenizer = lambda x: x.split(" "), binary='true').fit(y_train)
y_train = vectorizer.transform(y_train)
y_test = vectorizer.transform(y_test)


#topics + plots
xt_topic_plot_train=hstack([xt_train_multilabel,xtk_train_multilabel])
xt_topic_plot_test=hstack([xt_test_multilabel,xtk_test_multilabel])
```

In [21]:

```
#Randomsearch_Cv using Logisitic regression
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}
clf_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
RS_clf = RandomizedSearchCV(estimator=clf_estimator, param_distributions=params, n_iter=10, cv=5, s
coring='f1_micro', n_jobs=-1, verbose=10)
RS_clf.fit(xt_topic_plot_train, y_train)
print('Best estimator: ',RS_clf.best_estimator_)
print('Best Cross Validation Score: ',RS_clf.best_score_)

classifier2 = RS_clf.best_estimator_
classifier2.fit(xt_topic_plot_train, y_train)
predictions = classifier2.predict(xt_topic_plot_test)

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

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

print("Micro-average :")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   52.2s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  3.5min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 10.4min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 14.8min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 16.2min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 18.9min remaining:  4.2min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 24.2min remaining:  1.5min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 24.4min finished
```

```
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced',
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, 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:  0.33367728723098383
Accuracy : 0.029441624365482234
Hamming loss  0.06874002049522175
Micro-average :
Precision: 0.2925, Recall: 0.4235, F1-measure: 0.3460
```

In [25]:

```
#Computing Grid/Random search is computational expensive on BOW Vectorizer so we trying on TFIDF a
ccoring to research paper
#TFIDF N GRAMS(1,2)
#plots + keywords
#keywords
vectorizer_3 = 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))
xt_train_multilabel = vectorizer_3.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_3.transform(test_data['pre_pro_plot_synopsis'])

#topics
vectorizer_4 = TfidfVectorizer(smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), subl
inear_tf=False,
                             ngram_range=(1,1))
xtk_train_multilabel = vectorizer_4.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_4.transform(test_data['Keywords'])

#topics + plots

xt_topic_plot_train=hstack([xt_train_multilabel,xtk_train_multilabel])
xt_topic_plot_test=hstack([xt_test_multilabel,xtk_test_multilabel])

#Randomsearch_Cv using Logisitic regression
alpha = [0.0001,0.001,0.01,0.1,1,10,100,1000]
penalty = ['l1','l2']

params  = {'estimator__C': alpha,
          'estimator__penalty': penalty}
clf_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
RS_clf = RandomizedSearchCV(estimator=clf_estimator, param_distributions=params, n_iter=10, cv=5, s
coring='f1_micro', n_jobs=-1, verbose=10)
RS_clf.fit(xt_topic_plot_train, y_train)
print('Best estimator: ',RS_clf.best_estimator_)
print('Best Cross Validation Score: ',RS_clf.best_score_)

classifier2 = RS_clf.best_estimator_
classifier2.fit(xt_topic_plot_train, y_train)
predictions = classifier2.predict(xt_topic_plot_test)

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

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

print("Micro-average :")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  1.4min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  4.8min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 10.2min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 16.1min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 22.9min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 25.2min remaining:  5.5min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 25.9min remaining:  1.7min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 26.1min finished
```

```
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced',
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, 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:  0.3406132820133223
Accuracy : 0.031810490693739424
Hamming loss  0.06645694811849098
Micro-average :
Precision: 0.3033, Recall: 0.4224, F1-measure: 0.3531
```

In [26]:

```
#Computing Grid/Random search is computational expensive on BOW Vectorizer so we trying on TFIDF a
ccoring to research paper
#TFIDF N GRAMS(1,3)
#plots + keywords
#keywords
vectorizer_5 = 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))
xt_train_multilabel = vectorizer_5.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_5.transform(test_data['pre_pro_plot_synopsis'])

#topics
vectorizer_6 = TfidfVectorizer(smooth_idf=True, norm="l2", tokenizer = lambda x: x.split(" "), subl
inear_tf=False,
                             ngram_range=(1,1))
xtk_train_multilabel = vectorizer_6.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_6.transform(test_data['Keywords'])

#topics + plots
xt_topic_plot_train=hstack([xt_train_multilabel,xtk_train_multilabel])
xt_topic_plot_test=hstack([xt_test_multilabel,xtk_test_multilabel])

#Randomsearch_Cv using Logisitic regression
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}
clf_estimator = OneVsRestClassifier(LogisticRegression(class_weight='balanced'), n_jobs=-1)
RS_clf = RandomizedSearchCV(estimator=clf_estimator, param_distributions=params, n_iter=10, cv=5, s
coring='f1_micro', n_jobs=-1, verbose=10)
RS_clf.fit(xt_topic_plot_train, y_train)
print('Best estimator: ',RS_clf.best_estimator_)
print('Best Cross Validation Score: ',RS_clf.best_score_)

classifier2 = RS_clf.best_estimator_
classifier2.fit(xt_topic_plot_train, y_train)
predictions = classifier2.predict(xt_topic_plot_test)

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

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

print("Micro-average :")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision, recall, f1))
```

```
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:   47.1s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  3.4min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:  8.2min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 13.5min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 16.8min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 19.5min remaining:  4.3min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 23.3min remaining:  1.5min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 23.8min finished
```

```
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=1, class_weight='balanced',
                                                                 dual=False, fit_intercept=True,
                                                                 intercept_scaling=1,
                                                                 l1_ratio=None, 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:  0.33779041496260004
Accuracy : 0.028764805414551606
Hamming loss  0.06827768642310716
Micro-average :
Precision: 0.2971, Recall: 0.4322, F1-measure: 0.3521
```

In [30]:

```
#Computing Grid/Random search is computational expensive on BOW Vectorizer so we trying on TFIDF a
ccoring to research paper
#TFIDF UNI GRAMS
#plots + keywords
#keywords

vectorizer_1 = TfidfVectorizer(min_df=0.00009,max_features=10000,ngram_range=(1,1))
xt_train_multilabel = vectorizer_1.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_1.transform(test_data['pre_pro_plot_synopsis'])


#topics
vectorizer_2 = TfidfVectorizer(ngram_range=(1,1))
xtk_train_multilabel = vectorizer_2.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_2.transform(test_data['Keywords'])

#topics + plots
train_uni=hstack([xt_train_multilabel,xtk_train_multilabel])
test_uni=hstack([xt_test_multilabel,xtk_test_multilabel])
```

In [31]:

```
vectorizer_3 = TfidfVectorizer(min_df=0.00009,max_features=10000,ngram_range=(1,2))
xt_train_multilabel = vectorizer_3.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_3.transform(test_data['pre_pro_plot_synopsis'])

#topics
vectorizer_4 = TfidfVectorizer(ngram_range=(1,1))
xtk_train_multilabel = vectorizer_4.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_4.transform(test_data['Keywords'])

#topics + plots

train_bi=hstack([xt_train_multilabel,xtk_train_multilabel])
test_bi=hstack([xt_test_multilabel,xtk_test_multilabel])
```

In [32]:

```
vectorizer_5 = TfidfVectorizer(min_df=0.00009,max_features=10000,ngram_range=(1,3))
xt_train_multilabel = vectorizer_5.fit_transform(train_data['pre_pro_plot_synopsis'])
xt_test_multilabel = vectorizer_5.transform(test_data['pre_pro_plot_synopsis'])

#topics
vectorizer_6 = TfidfVectorizer(ngram_range=(1,1))
xtk_train_multilabel = vectorizer_6.fit_transform(train_data['Keywords'])
xtk_test_multilabel = vectorizer_6.transform(test_data['Keywords'])

#topics + plots
train_tri=hstack([xt_train_multilabel,xtk_train_multilabel])
test_tri=hstack([xt_test_multilabel,xtk_test_multilabel])
```

In [33]:

```
#uni + bi
x_train_uni_bi = hstack([train_uni,train_bi])
x_test_uni_bi = hstack([test_uni,test_bi])
```

In [34]:

```
x_train_uni_bi.shape
```

Out[34]:

```
(11797, 20380)
```

In [35]:

```
#uni + bi + tri
x_train_uni_bi_tri = hstack([x_train_uni_bi,train_tri])
x_test_uni_bi_tri = hstack([x_test_uni_bi,test_tri])
```

In [36]:

```
x_train_uni_bi_tri.shape
```

Out[36]:

```
(11797, 30570)
```

In [39]:

```
#uni bi tri
vectorizer_1_6 = CountVectorizer(tokenizer = lambda x: x.split(" "), binary='true').fit(y_train)
y_train_6 = vectorizer_1_6.transform(y_train)
y_test_6 = vectorizer_1_6.transform(y_test)
print('model Started......!')

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

params  = {'estimator__C': alpha}
clf_estimator_6 =
OneVsRestClassifier(LogisticRegression(class_weight='balanced',penalty='l2',n_jobs=-1),n_jobs=-1)
RS_clf_6 = RandomizedSearchCV(estimator=clf_estimator_6, param_distributions=params, n_iter=10, cv=
5, scoring='f1_micro', n_jobs=-1,verbose=10)
RS_clf_6.fit(x_train_uni_bi_tri, y_train_6)
print('Best estimator: ',RS_clf_6.best_estimator_)
print('Best Cross Validation Score: ',RS_clf_6.best_score_)

classifier_6 = RS_clf_6.best_estimator_
classifier_6.fit(x_train_uni_bi_tri, y_train_6)
predictions_6 = classifier_6.predict(x_test_uni_bi_tri)

print("Accuracy :",metrics.accuracy_score(y_test_6, predictions_6))
print("Hamming loss ",metrics.hamming_loss(y_test_6,predictions_6))

precision_6 = precision_score(y_test_6, predictions_6, average='micro')
recall_6 = recall_score(y_test_6, predictions_6, average='micro')
f1_6 = f1_score(y_test_6, predictions_6, average='micro')

print("Micro-average :")
print("Precision: {:.4f}, Recall: {:.4f}, F1-measure: {:.4f}".format(precision_6, recall_6, f1_6))
print('Model Ended......!')
```

```
model Started......!
Fitting 5 folds for each of 10 candidates, totalling 50 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:  3.3min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:  8.8min
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed: 16.2min
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed: 32.4min
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed: 47.3min
[Parallel(n_jobs=-1)]: Done  41 out of  50 | elapsed: 74.3min remaining: 16.3min
[Parallel(n_jobs=-1)]: Done  47 out of  50 | elapsed: 93.8min remaining:  6.0min
[Parallel(n_jobs=-1)]: Done  50 out of  50 | elapsed: 97.7min finished
```

```
Best estimator:  OneVsRestClassifier(estimator=LogisticRegression(C=0.9, class_weight='balanced',
                                                 dual=False, fit_intercept=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=100,
```

```
                                                  multi_class='warn', n_jobs=-1,
                                                  penalty='l2',
                                                  random_state=None,
                                                  solver='warn', tol=0.0001,
                                                  verbose=0, warm_start=False),
                    n_jobs=-1)
Best Cross Validation Score:  0.3250946815542577
Accuracy : 0.02673434856175973
Hamming loss  0.07082290698505755
Micro-average :
Precision: 0.2834, Recall: 0.4250, F1-measure: 0.3400
Model Ended......!
```

Deep learning LSTM

In [86]:

```
#data
tr_ip1= train_data['pre_pro_plot_synopsis']
te_ip1= test_data['pre_pro_plot_synopsis']

# prepare tokenizer for train
tr = Tokenizer(num_words=10000)
tr.fit_on_texts(tr_ip1)
vocab_size = len(tr.word_index) + 1
# integer encode the documents in train
encoded_tr_ip1 = tr.texts_to_sequences(tr_ip1)
# integer encode the documents in test
encoded_te_ip1 = tr.texts_to_sequences(te_ip1)
```

In [76]:

```
#!wget http://nlp.stanford.edu/data/glove.6B.zip
```

In [77]:

```
#!unzip glove*.zip
```

In [87]:

```
#pad documents to a max length of max words
max_seq_length = 500 # max length of a pre-processed essay
ip_1_train = pad_sequences(encoded_tr_ip1, maxlen=max_seq_length)
print('shape of ip_1_train',ip_1_train.shape)
ip_1_test = pad_sequences(encoded_te_ip1, maxlen=max_seq_length)
print('shape of ip_1_test',ip_1_test.shape)
```

```
shape of ip_1_train (11797, 500)
shape of ip_1_test (2955, 500)
```

In [88]:

```
# load the whole embedding into memory
embeddings_index = dict()
f = open('glove.6B.300d.txt')
for line in f:
    values = line.split()
    word = values[0]
    coefs = np.asarray(values[1:], dtype='float32')
    embeddings_index[word] = coefs
f.close()
```

In [89]:

```
embedding_matrix = np.zeros((vocab_size, 300))
for word, i in tr.word_index.items():
    embedding_vector = embeddings_index.get(word)
    if embedding_vector is not None:
        embedding_matrix[i] = embedding_vector
embedding_matrix.shape
```

```
embedding_matrix.shape
```

Out[89]:

```
(111892, 300)
```

In [46]:

```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(" "), binary='true').fit(y_train)
y_train = vectorizer.transform(y_train)
y_test = vectorizer.transform(y_test)
print(y_train.shape)
```

```
(11797, 71)
```

In [102]:

```
# create the model
model = Sequential()
model.add(Embedding(vocab_size,300, weights=[embedding_matrix],trainable=False,
input_length=max_seq_length))
model.add(Dropout(0.2))
model.add(Conv1D(128,5,activation='relu'))
model.add(MaxPooling1D(pool_size=4))
model.add(LSTM(100))
model.add(Dropout(0.2))
model.add(BatchNormalization())
model.add(Dense(71, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
print(model.summary())

model.fit(ip_1_train, y_train, nb_epoch=10, batch_size=256,validation_data=(ip_1_test,y_test))
```

```
Model: "sequential_19"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_23 (Embedding)     (None, 500, 300)          33567600
_________________________________________________________________
dropout_23 (Dropout)         (None, 500, 300)          0
_________________________________________________________________
conv1d_14 (Conv1D)           (None, 496, 128)          192128
_________________________________________________________________
max_pooling1d_13 (MaxPooling (None, 124, 128)          0
_________________________________________________________________
lstm_23 (LSTM)               (None, 100)               91600
_________________________________________________________________
dropout_24 (Dropout)         (None, 100)               0
_________________________________________________________________
batch_normalization_23 (Batc (None, 100)               400
_________________________________________________________________
dense_20 (Dense)             (None, 71)                7171
=================================================================
Total params: 33,858,899
Trainable params: 291,099
Non-trainable params: 33,567,800
_________________________________________________________________
None
Train on 11797 samples, validate on 2955 samples
Epoch 1/10
11797/11797 [==============================] - 13s 1ms/step - loss: 13.0214 - accuracy: 0.1044 - v
al_loss: 11.5835 - val_accuracy: 0.1723
Epoch 2/10
11797/11797 [==============================] - 10s 852us/step - loss: 13.7986 - accuracy: 0.1499 -
val_loss: 12.1839 - val_accuracy: 0.1530
Epoch 3/10
11797/11797 [==============================] - 10s 825us/step - loss: 16.1579 - accuracy: 0.1494 -
val_loss: 15.0776 - val_accuracy: 0.1330
Epoch 4/10
11797/11797 [==============================] - 10s 842us/step - loss: 19.0781 - accuracy: 0.1313 -
val_loss: 15.0543 - val_accuracy: 0.1371
Epoch 5/10
11797/11797 [==============================] - 10s 853us/step - loss: 22.0941 - accuracy: 0.1319 -
val_loss: 21.0597 - val_accuracy: 0.1577
```

```
Epoch 6/10
11797/11797 [==============================] - 10s 835us/step - loss: 25.0698 - accuracy: 0.1327 -
val_loss: 18.2127 - val_accuracy: 0.1692
Epoch 7/10
11797/11797 [==============================] - 10s 865us/step - loss: 28.8377 - accuracy: 0.1266 -
val_loss: 27.5180 - val_accuracy: 0.1479
Epoch 8/10
11797/11797 [==============================] - 10s 837us/step - loss: 34.0514 - accuracy: 0.1128 -
val_loss: 30.6786 - val_accuracy: 0.1557
Epoch 9/10
11797/11797 [==============================] - 10s 838us/step - loss: 40.1147 - accuracy: 0.1113 -
val_loss: 23.5263 - val_accuracy: 0.1100
Epoch 10/10
11797/11797 [==============================] - 10s 837us/step - loss: 46.9542 - accuracy: 0.1045 -
val_loss: 28.6237 - val_accuracy: 0.1486
```

Out[102]:

```
<keras.callbacks.callbacks.History at 0x7fb76e463898>
```

In [103]:

```python
# Final evaluation of the model on test data
scores = model.evaluate(ip_1_test, y_test, verbose=0)
print('Test loss:', scores[0])
print('Test accuracy:', scores[1])
```

```
Test loss: 28.623673683334324
Test accuracy: 0.1485617607831955
```

In [5]:

```python
from prettytable import PrettyTable
x = PrettyTable()
print('==========Models computed using Simple loop hyperparameter tuning...==========')
x.field_names = [ "Model","vectorizer", "best-alpha", "Precision ","Recall","F1-score"]
x.add_row(["LOGISTIC REGR","TFIDF-UNI",      0.1,0.27,0.38,0.32])
x.add_row(["LOGISTIC REGR","TFIDF-BI",      1.0,0.23,0.45,0.31])
print(x)

x2 = PrettyTable()
print('==========Models computed using Randomsearch hyperparameter tuning...==========')
x2.field_names = [ "Model","vectorizer", "best-alpha", "Precision ","Recall","F1-score"]
x2.add_row(["LOGISTIC REGR","TFIDF-UNI",      1.0,0.29,0.42,0.34])
x2.add_row(["LOGISTIC REGR","TFIDF-BI",      1.0,0.30,0.42,0.35])
x2.add_row(["LOGISTIC REGR","TFIDF-TRI",      1.0,0.29,0.43,0.35])
x2.add_row(["LOGISTIC REGR","TFIDF-UNI+BI+TRI",      0.9,0.28,0.42,0.34])
print(x2)
```

```
==========Models computed using Simple loop hyperparameter tuning...==========
+---------------+------------+------------+------------+--------+----------+
|     Model     | vectorizer | best-alpha | Precision  | Recall | F1-score |
+---------------+------------+------------+------------+--------+----------+
| LOGISTIC REGR | TFIDF-UNI  |    0.1     |    0.27    |  0.38  |   0.32   |
| LOGISTIC REGR |  TFIDF-BI  |    1.0     |    0.23    |  0.45  |   0.31   |
+---------------+------------+------------+------------+--------+----------+
==========Models computed using Randomsearch hyperparameter tuning...==========
+---------------+------------------+------------+------------+--------+----------+
|     Model     |    vectorizer    | best-alpha | Precision  | Recall | F1-score |
+---------------+------------------+------------+------------+--------+----------+
| LOGISTIC REGR |    TFIDF-UNI     |    1.0     |    0.29    |  0.42  |   0.34   |
| LOGISTIC REGR |     TFIDF-BI     |    1.0     |    0.3     |  0.42  |   0.35   |
| LOGISTIC REGR |    TFIDF-TRI     |    1.0     |    0.29    |  0.43  |   0.35   |
| LOGISTIC REGR | TFIDF-UNI+BI+TRI |    0.9     |    0.28    |  0.42  |   0.34   |
+---------------+------------------+------------+------------+--------+----------+
```

Observations

1. Applied Feature Extraction Topic Modelling to improve the F1-score We got 0.35 for TFIDF Tri grams and BI grams
2. Applied Deep learning algorithm LSTM it is not performs good and noticed if we increase the no of layers it crashing the system memory looks like requrie huge computational resources

system memory looks like require huge computational resources

3. In previous models we got 0.37 as highest F1-score using 71-Tags by using topic modelling we got 0.35 as F1-score and usig Top-3 and Top-5 Tags we got 0.58 as highest F1-score using Uni-Grams