# MPST : Tag Prediction

### Library Imports

In [0]:

```
import warnings
import spacy
import os
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
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 *
from sklearn import metrics
from sklearn.metrics import *
from sklearn import svm
from sklearn.naive_bayes import *
from skmultilearn.adapt import mlknn
from skmultilearn.problem_transform import ClassifierChain
from skmultilearn.problem_transform import BinaryRelevance
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
from nltk.util import skipgrams
import nltk
from nltk import pos_tag, word_tokenize
from nltk.sentiment.vader import SentimentIntensityAnalyzer
from gensim.models import Word2Vec
import pickle
from collections import Counter
from scipy.sparse import hstack
from scipy.sparse import save_npz, load_npz
from sklearn.preprocessing import *
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from numpy import array
from numpy import asarray
from numpy import zeros
from prettytable import PrettyTable
from tensorflow.keras.models import *
from tensorflow.keras.layers import *
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.callbacks import *
from tensorflow.keras import regularizers
from tensorflow.keras.optimizers import *
```

In [4]:

```
from google.colab import drive
drive.mount('/content/drive', force_remount=True)

drivePath = '/content/drive/My Drive/Colab Notebooks/'
#drivePath = './'
```

```
Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?cli
ent_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleuserconten
t.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&
scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%
3a%2f%2fwww.googleapis.com%2fauth%2fdrive%20https%3a%2f%2fwww.googleapis.c
om%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2faut
h%2fpeopleapi.readonly

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

# MPST: Movie Plot Synopses with Tags

A dataset of movie plot synopses with story related tags

# 1. Business Problem

## 1.1 Description

Abstract Social tagging of movies reveals a wide range of heterogeneous information about movies, like the genre, plot structure, soundtracks, metadata, visual and emotional experiences. Such information can be valuable in building automatic systems to create tags for movies. Automatic tagging systems can help recommendation engines to improve the retrieval of similar movies as well as help viewers to know what to expect from a movie in advance. In this paper, we set out to the task of collecting a corpus of movie plot synopses and tags. We describe a methodology that enabled us to build a fine-grained set of around 70 tags exposing heterogeneous characteristics of movie plots and the multi-label associations of these tags with some 14K movie plot synopses. We investigate how these tags correlate with movies and the flow of emotions throughout different types of movies. Finally, we use this corpus to explore the feasibility of inferring tags from plot synopses. We expect the corpus will be useful in other tasks where analysis of narratives is relevant.

## Problem Statemtent

Suggest the tags based on the content that was there in the Movie in its Title and Synopsis.

### 1.2 Useful Links :

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

Research Paper : https://www.aclweb.org/anthology/L18-1274/ (https://www.aclweb.org/anthology/L18-1274/)

Additional Info : http://ritual.uh.edu/mpst-2018/ (http://ritual.uh.edu/mpst-2018/)

### 1.3 Real World / Business Objectives and Constraints

1. Predict as many tags as possible with high precision and recall.
2. Incorrect tags could impact customer experience on StackOverflow.
3. No strict latency constraints.

## 2. Machine Learning problem

### 2.1 Data

#### 2.1.1 Data Overview

All of the data is in 1 file: mpst_full_data.csv

```
Data is divided into 3 Sections Train, Test and Val data within the file
Train Consist of 64% of the data.

Test Consist of 20% of the data.

Val Consist of 16% of the data.

Number of rows in mpst_full_data.csv = 14828
```

**Data Field Explaination**

**Dataset contains 14,828 rows. The columns in the table are:**

```
imdb_id - IMDB id of the movie

title - Title of the movie

plot_synopsis - Plot Synopsis of the movie

Tags - Tags assigned to the movie separated by ","

split - Position of the movie in the standard data split, like Train, Test or Va
l

synopsis_source - From where the plot synopsis was collected
```

### 2.1.2 Example Data point

```
imdb_id:  tt0113862
title : Mr. Holland's Opus
plot_synopsis:

Glenn Holland, not a morning person by anyone's standards, is woken up by his wi
fe Iris early one bright September morning in 1964.
 Glenn has taken a job as a music teacher at the newly renamed John F.
 Kennedy High School.
 He intends his job to be a sabbatical from being a touring musician, during whi
ch he hopes to have more "free time" to compose.
 However, he soon finds that his job as a teacher is more time-consuming than he
first thought.
As he arrives at the school for the first time, he meets Vice Principal Wolters,
who comments on his Corvair,
the model of car the Ralph Nader wrote a book about.
 Inside the building, he meets Principal Helen Jacobs.
 Having got off to an awkward start with both of them, he goes to the music room
and meets his students for the first time.
 The students are dull, apathetic, and mostly terrible musicians.
 At lunchtime, he meets the football coach, Bill Meister, and strikes up a frien
dship with him.
 At the end of his stressful first day, Glenn and Iris talk about their future.
 If everything goes according to plan, between his paychecks and what she made w
ith her photography, he should be able to quit in four
 years and go back to his music, including composing.
Glenn notices one dedicated but inept clarinet player, Gertrude Lang, and starts
working with her individually.
 He continues attempting to teach the class about music and continues working on
his music at home as time passes.
 Grading papers gradually replaces working on his own music during his home tim
e, much to his chagrin.
 After several months, Glenn grows exasperated when it seems that none of his st
udents have learned anything from his classes.
 Gertrude, despite diligent practice, does not improve her clarinet-playing.
 Glenn's exasperation is further compounded when the Principal Jacobs chastises
 him for not focusing properly on his students.
 She has noticed that he is even happier to leave at the end of each day than mo
st of the students.
 Later, Glenn expounds his frustration to Iris, who then informs him that she's
 pregnant.
 Glenn is dumbstruck, and his muteness upsets Iris.
 To comfort her, he tells her a story about how he discovered John Coltrane (his
favorite musician) records as a teenager,
 the point being he could get used to this turn of affairs.
After some soul-searching, Glenn decides to try some unconventional methods of t
eaching music appreciation, including the use of
'Rock and Roll' to interest students, demonstrating to them the similarities bet
ween Bach's "Minuet in G" and rock-and-roll in the
form of the Toys' "Lovers Concerto".
 For the first time, the students are interested in the class, and Glenn appears
much happier as he relates this to Iris as they
```

assemble a crib.
Their apartment is getting more and more crowded, and Glenn suggests that they get a house.
Iris is overjoyed, even though it means using their savings and Glenn sacrificing his summer vacation, which he intended to use to
work on his composing, in order to make extra money teaching Driver's Ed.
Glenn does right by his family but he knows he can forget about getting out of the teaching gig for the foreseeable future.
Continuing his new, unorthodox teaching methods, he finally gets Gertrude, who was on the verge of giving up, to have a breakthrough
and become a more skilled clarinet player.
She rediscovers her joy of playing, and the now-competent band go on to play at the 1965 graduation.
Summer vacation begins, and Glenn follows through on his plan to teach Driver's Ed, having a series of near-death experiences at the
hands of new drivers.
Glenn and Iris move into their new house.
Soon, we see the Driver's Ed car once again, except this time it is Glenn himself driving like a maniac, breaking every traffic law -
so that he could get to the hospital to see his newborn son, Coltrane ("Cole") Holland.
Glenn's unorthodox teaching methods do not go unnoticed by Principal Jacobs, or Vice-Principal Wolters.
They, along with the conservative School Board and the parents of the community, are hostile to rock-and-roll.
Glenn is able to convince the principal that he believes strongly that teaching the students about all music, including rock-and-roll,
will help them appreciate it all the more.
The principal and vice principal also hand him a new assignment, to get a marching band together for the football team.
Glenn is at a loss with this concept, until his Bill Meister agrees to help, in exchange for Glenn putting one of his football players,
Louis Russ, in the band to allow him to earn an extra curricular activity credit, which he needs in order to stay eligible for the sports
teams.
Louis knows absolutely nothing about music, but takes up drums.
He has trouble keeping time and always finds himself out of place with the rest of the band.
Later, Glenn and Bill are chatting while playing chess.
Bill, a bachelor, wants to know about Glenn's stories of debauchery as a traveling musician, but Glenn doesn't want to talk about the past,
as he is a different person in a different time.
Glenn instead tells Bill he is pessimistic about Louis Russ.
Bill encourages him to keep trying.
Much as he worked with Gertrude earlier, Glenn starts working one-on-one with Louis, helping to get a feel for the tempo of music.
After some hard work, Louis gets it, and later, he marches with the band in the local parade, much to the delight of his family.
Immediately behind the Kennedy band in the parade is a fire engine, and its deafeningly loud horn catches everyone by surprise.
Iris looks into Cole's stroller to check on him, but the noise hasn't awakened Cole - the boy is deaf.

The revelation drives a wedge between Glenn and his son, as it seems that his s
on cannot understand what he does.
A more somber Glenn teaches his students about Beethoven, the deaf composer.
Time passes, and we see a montage of events from the late '60s, as Glenn picks a
way as his composition a little at a time and watches
Iris work with Cole.
We stop again in the early '70s, with Glenn still directing his high school ban
d.
Cole is old enough to enter school.
Because of her mounting frustration with her inability to communicate with Col
e, she insists on sending him to a special school for the
deaf, whatever the cost.
The three of them visit the school.
Glenn winces at the cost, but they enroll Cole and set about to learn sign lang
uage themselves, though Iris puts more effort into it than
Glenn.
Apathetic students still go through Glenn's classes, and one of them, named Stad
ler, is stoned.
Glenn is chewing him out when Glenn receives bad news.
He tells Stadler to meet him on Saturday.
On that day, they appear at a funeral.
Louis Russ has come home from Vietnam after being killed in action.
Coach Meister is there, and he and Glenn mourn.
At the end of that academic year, Bill reveals that he finally has a steady gir
l-friend, and Principal Jacobs retires from the high school,
praising Glenn for what he has done.
We see another montage of events, this time in the 1970s.
Glenn continues teaching Driver's Ed in the summer.
We see the class of 1980 being welcomed back, suggesting that it is now Septemb
er 1979.
Glenn and Bill Meister team up to help the Drama Department, when it is rumored
that funding may be pulled.
Glenn and Bill tell Wolters, now the principal, have an idea to be certain the
school will make money rather than lose it; it will be a
musical revue of Gershwin classics.
During auditions for the musical revue, Glenn becomes entranced and interested
in a talented young singer named Rowena Morgan.
At home, the teenage Cole comes home and tells Glenn about the science fair, wh
ich Glenn missed.
Iris is fluent at sign language now, but Glenn is still only fair.
Iris reproaches him for spending so much time with the school projects and the
students while neglecting his own son.
Glenn is frustrated, realizing that his own musical composing has been on the b
ack burner for 15 years now.
Rowena visits Glenn at a diner, where he has gotten into the habit of going to g
et out of the house and have someplace quiet to work.
Unknown to Iris, Glenn writes a small piece that he titles "Rowena's Theme," an
d takes an interest when Rowena states that she wants to
leave town and go to New York to sing professionally.
Glenn's life at home is still strained.
Iris agrees to come to the school play on Saturday, because she had a meeting w
ith Cole's teachers on opening night (Friday).

The school revue arrives at last, and is a big hit, playing to a packed house.
 In the audience we see Coach Meister wearing a ring (he married the woman we sa
w earlier), and Sarah, the drama teacher, shows Principal
 Wolters something on a new invention, a handheld calculator (presumably, showin
g him how much gate money made it into the school coffers,
 as Wolters looks impressed).
 After the revue, Rowena comes to see Glenn in the auditorium, and she tells him
she intends to pursue her dream of singing by going to
 New York the very next night, after the second and last performance of the revu
e.
 Glenn is taken aback.
 Rowena hints that she'd like Glenn to come with her.
 Glenn goes home and looks at his photo album, looking at pictures of his family
and pictures of his old life as a traveling musician,
 now half a lifetime in the past.
 He is tempted to leave everything behind and go with Rowena to restore his old
 life as a musician.
 However, he realizes he is no longer the same person as he was then.
 He visits Rowena at the bus stop and sees her off, giving her the names of some
one in New York who will help her find lodging.
 Glenn watches her depart, and goes home, content in his love for Iris.
The timeline then shifts to late 1980, when John Lennon is killed.
 Glenn goes home and finds Cole working on Glenn's old Corvair.
 When Cole asks what is wrong, Glenn tries to explain, but then gives up, feelin
g that his son wouldn't understand John Lennon or his music.
 This infuriates Cole who (through Iris), explains that he does care about Glenn
and knows about John Lennon, but that Glenn does not seem
 to be at all interested in communicating with him.
 Cole berates his father for putting so much effort in to teaching his students
 and very little towards him, calling him an asshole in sign
 language as he stalks off.
 Glenn then makes an effort, and even provides a concert at the high school, whi
ch also features lights and other items to enhance the show
 for deaf members of the school where Cole attends.
 Glenn, having become somewhat more proficient in sign language, even does an in
terpretation of Lennon's song 'Beautiful Boy,' dedicated to
 Cole.
 Later, Glenn discovers Cole listening to records by sitting on the speakers and
feeling the vibrations through his body, and they can start
 healing the rift between them, even as Glenn's composition continues to gather
 dust.
Time passes.
 It's now 1995.
 Glenn goes to see Principal Wolters, who announces that Art, Music and Drama ha
ve been cut from the school curriculum, and Glenn would be
 out of a job shortly.
 Glenn, who has become a cynical old man, tells Wolters that to cut the fine art
s would lead to a generation of students who would be
 proficient at reading and writing and math (maybe) but would have nothing to re
ad or write about.
 Wolters offers to write Glenn a reference, but Glenn, who is now 60 years old,
 fully recognizes the futility of the gesture.

```
 His working days are over and he knows it.
 Then he looks up at the picture on the wall of the long-departed former Princip
al Jacobs.
 He says Jacobs would have fought the budget cuts, and he will too.
 Glenn pleads to the school board to reconsider, but they refuse.
At home, Iris reads a letter from the now-grown Cole.
 He has become a teacher himself, and was considering an offer from a university
for the deaf in Washington, D.
C.
 He also has taken Glenn's old car, the Corvair that we saw at the beginning and
that Cole was working on in his teens, and jokingly
 writes that he will never give it back.
 Despondent, Glenn walks through the school on his last day, and he talks to Coa
ch Bill, whose job as football coach is safe, though
 he can't be far from retirement himself.
 Glenn figures that he will bring in some money teaching piano lessons on the si
de, but he's unprepared to be forced into early retirement.
On Glenn's final day at the school, Cole shows up driving the Corvair.
 School's out for him, too.
 Glenn is surprised when Iris and Cole lead him to the school auditorium, where
 they have organized a surprise going-away celebration for
 him.
 He sees many of his former students in the audience, including Stadler, the pot
head from years before.
 Arriving next is Gertrude Lang, the clarinetist who Glenn helped in the 60s, wh
o has since become the state's governor.
 Gertrude thanks Glenn for his dedication, and Glenn is very moved.
 He is moved to tears when she gives him a baton and asks him to conduct his own
composition, which she had got hold of.
 The curtains open and a band, filled with more of Glenn's former students, is a
ssembled and ready to play.
 Governor Lang picks up her clarinet and takes her place among them, and they pl
ay, for the first time, the musical Opus that Glenn had
 been picking away at for three decades.

tags : inspiring, romantic, stupid, feel-good
split : train
synopsis_source: imdb
```

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

### 2.2.1 Type of Machine Learning Problem

**It is a multi-label classification problem**
**Multi-label Classification: Multilabel classification assigns to each sample a set of target labels. This can be thought as predicting properties of a data-point that are not mutually exclusive, such as topics that are relevant for a document. A Movie might be about any Genre like romantic, action, thriller, horror at the same time or none of these.**
**__Credit__: http://scikit-learn.org/stable/modules/multiclass.html**

### 2.2.2 Performance metric

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

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

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

**'Micro f1 score':**
**Calculate metrics globally by counting the total true positives, false negatives and false positives. This is a better metric when we have class imbalance.**

**'Macro f1 score':**
**Calculate metrics for each label, and find their unweighted mean. This does not take label imbalance into account.**

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

**Hamming loss : The Hamming loss is the fraction of labels that are incorrectly predicted.**
**https://www.kaggle.com/wiki/HammingLoss (https://www.kaggle.com/wiki/HammingLoss)**

# 3. Exploratory Data Analysis

## 3.1 Data Loading and Cleaning

### 3.1.1 Using Pandas to Load the data

In [0]:

```
filePath = str(drivePath)+'MPST Dataset/mpst_full_data.csv'

data = pd.read_csv(filePath, header=0)
```

In [6]:

```
print("Number of rows in the whole dataset :",data.shape)
```

```
Number of rows in the whole dataset : (14828, 6)
```

In [7]:

```
# Select duplicate rows based on ID
ids = data["imdb_id"]
print("Duplicate Rows :")
data[ids.isin(ids[ids.duplicated()])].sort_values("imdb_id")
```

```
Duplicate Rows :
```

Out[7]:

| imdb_id | title | plot_synopsis | tags | split | synopsis_source |
|---|---|---|---|---|---|

In [8]:

```
data["tag_count"] = data["tags"].apply(lambda text: len(text.split(",")))
# adding a new feature number of tags per movie
data.head()
```

Out[8]:

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

In [9]:

```
print("No. of movies that doesnt have tags : ")
print(data[data["tag_count"]==0].shape[0])
```

```
No. of movies that doesnt have tags : 
0
```

In [10]:

```
print("Dropping rows with no tags : ")
init_rows = data.shape[0]
data.drop(data[data['tag_count'] == 0].index, inplace = True)
final_rows = data.shape[0]
print("Total rows dropped : ",(init_rows-final_rows))
```

```
Dropping rows with no tags :
Total rows dropped :  0
```

In [11]:

```
# distribution of number of tags per movie
data.tag_count.value_counts()
```

Out[11]:

```
1     5516
2     3124
3     1959
4     1238
5      916
6      606
7      478
8      316
9      223
10     135
11     107
13      52
12      51
14      33
15      28
16      16
18      11
17       9
21       3
23       2
20       2
19       2
25       1
Name: tag_count, dtype: int64
```

## 3.2 Analysis of Tags

### 3.2.1 Total number of unique tags

In [12]:

```
set_of_tags = set()
preprocessed_tags = list()

for row in data['tags']:
  for tag in row.split(","):
    new_tag = tag.strip()
    set_of_tags.add(new_tag)
  preprocessed_tags.append(row.strip().replace(",",""))

print(set_of_tags)
```

```
{'good versus evil', 'action', 'tragedy', 'cruelty', 'realism', 'brainwash
ing', 'satire', 'murder', 'historical', 'dark', 'atmospheric', 'autobiogra
phical', 'magical realism', 'feel-good', 'christian film', 'pornographic',
'melodrama', 'western', 'sci-fi', 'claustrophobic', 'absurd', 'comedy', 'c
omic', 'prank', 'plot twist', 'storytelling', 'violence', 'anti war', 'adu
lt comedy', 'non fiction', 'psychedelic', 'neo noir', 'paranormal', 'alter
nate history', 'sentimental', 'dramatic', 'grindhouse film', 'stupid', 'hu
mor', 'sadist', 'boring', 'historical fiction', 'thought-provoking', 'haun
ting', 'allegory', 'inspiring', 'suspenseful', 'suicidal', 'revenge', 'whi
msical', 'blaxploitation', 'horror', 'gothic', 'philosophical', 'romanti
c', 'home movie', 'insanity', 'clever', 'avant garde', 'intrigue', 'blea
k', 'mystery', 'psychological', 'queer', 'cult', 'cute', 'fantasy', 'depre
ssing', 'flashback', 'entertaining', 'alternate reality'}
```

In [13]:

```
print("Total Number of unique Tags : ",len(set_of_tags))
```

```
Total Number of unique Tags :  71
```

In [0]:

```
# Importing & Initializing the "CountVectorizer" object, which
#is scikit-learn's bag of words tool.

#by default 'split()' will tokenize each tag using space.
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(" "), vocabulary=set_of_tags
)
# fit_transform() does two functions: First, it fits the model
# and learns the vocabulary; second, it transforms our training data
# into feature vectors. The input to fit_transform should be a list of strings.
tag_dtm = vectorizer.fit_transform(preprocessed_tags)
```

In [15]:

```
print("Number of data points :", tag_dtm.shape[0])
print("Number of unique tags :", tag_dtm.shape[1])
```

```
Number of data points : 14828
Number of unique tags : 71
```

In [16]:

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

```
Some of the tags we have : ['absurd', 'action', 'adult comedy', 'allegor
y', 'alternate history', 'alternate reality', 'anti war', 'atmospheric',
'autobiographical', 'avant garde']
```

In [0]:

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

In [18]:

```
tag_df = pd.DataFrame(list(result.items()), columns=['Tags', 'Counts'])
tag_df.head()
```

Out[18]:

| | Tags | Counts |
|---|---|---|
| 0 | absurd | 270 |
| 1 | action | 664 |
| 2 | adult comedy | 0 |
| 3 | allegory | 139 |
| 4 | alternate history | 0 |

In [0]:

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

In [20]:

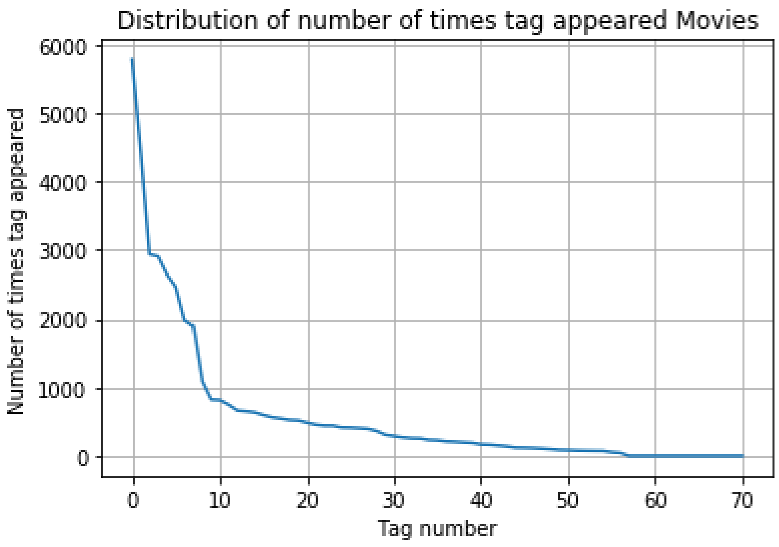
```
tag_df_sorted.head()
```

Out[20]:

| | Tags | Counts |
|---|---|---|
| 43 | murder | 5782 |
| 68 | violence | 4426 |
| 28 | flashback | 2937 |
| 57 | romantic | 2906 |
| 20 | cult | 2647 |

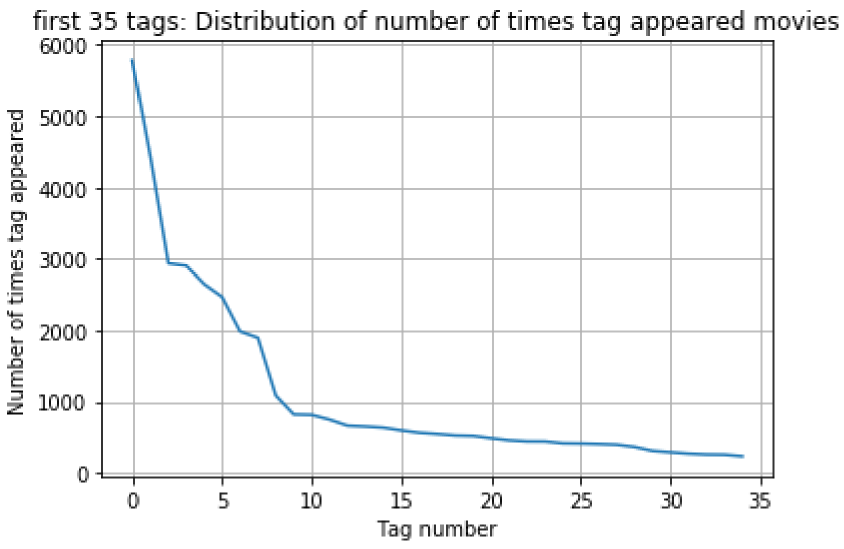
In [0]:

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



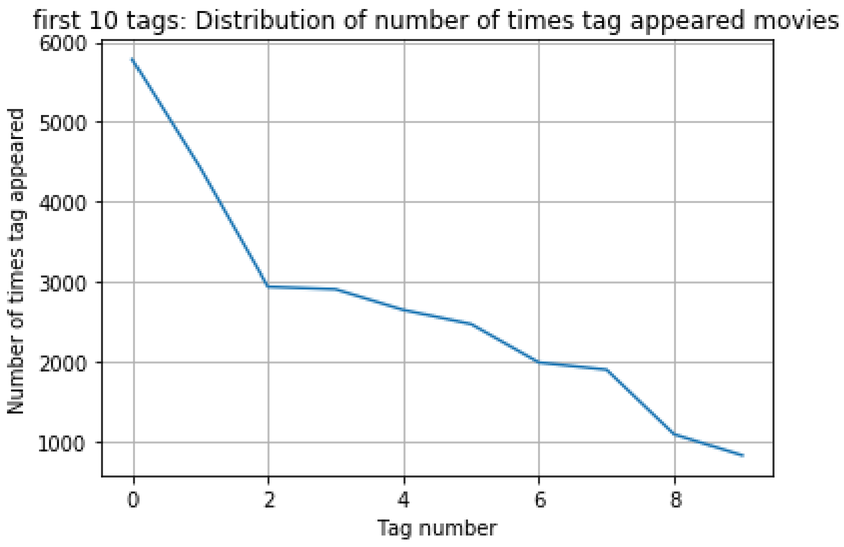
In [0]:

```
plt.plot(tag_counts[0:35])
plt.title('first 35 tags: Distribution of number of times tag appeared movies')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```



In [0]:

```
plt.plot(tag_counts[0:10])
plt.title('first 10 tags: Distribution of number of times tag appeared movies')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.show()
```

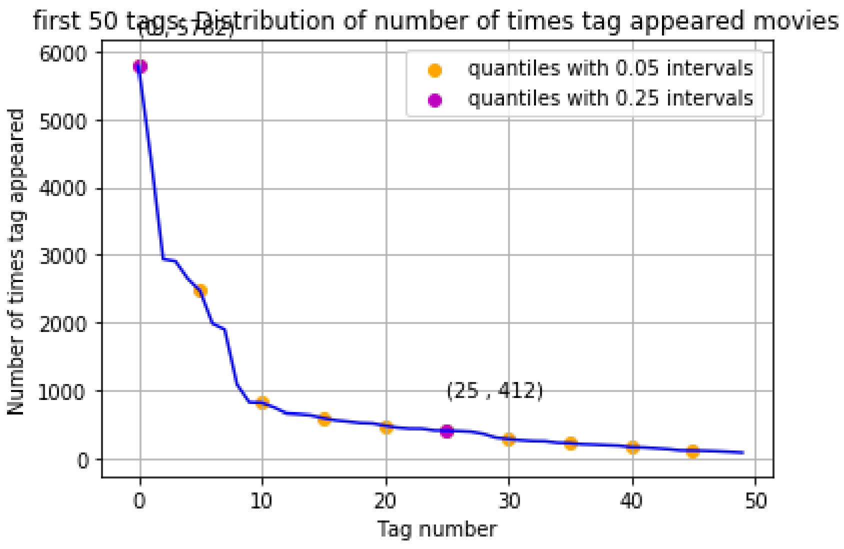


In [0]:

```
plt.plot(tag_counts[0:50], c='b')
plt.scatter(x=list(range(0,50,5)), y=tag_counts[0:50:5], c='orange', label="quantiles w
ith 0.05 intervals")
# quantiles with 0.25 difference
plt.scatter(x=list(range(0,50,25)), y=tag_counts[0:50:25], c='m', label = "quantiles wi
th 0.25 intervals")

for x,y in zip(list(range(0,50,25)), tag_counts[0:50:25]):
    plt.annotate(s="({} , {})".format(x,y), xy=(x,y), xytext=(x-0.05, y+500))

plt.title('first 50 tags: Distribution of number of times tag appeared movies')
plt.grid()
plt.xlabel("Tag number")
plt.ylabel("Number of times tag appeared")
plt.legend()
plt.show()
```



In [0]:

```
# Store tags greater than 10K in one list
lst_tags_gt_500 = tag_df[tag_df.Counts>500].Tags
#Print the length of the list
print ('{} Tags are used more than 500 times'.format(len(lst_tags_gt_500)))
# Store tags greater than 100K in one list
lst_tags_gt_1k = tag_df[tag_df.Counts>1000].Tags
#Print the length of the list.
print ('{} Tags are used more than 1000 times'.format(len(lst_tags_gt_1k)))
```

```
20 Tags are used more than 500 times
9 Tags are used more than 1000 times
```

**Observations:**

1. **There are total 20 tags which are used more than 500 times.**
2. **9 tags are used more than 1000 times.**
3. **Most frequent tag (i.e. murder) is used 5782 times.**
4. **Since some tags occur much more frequenctly than others, Micro-averaged F1-score is the appropriate metric for this probelm.**

### 3.2.4 Tags Per Movie

In [0]:

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

print(tag_quest_count[:5])
```

```
We have total 14828 datapoints.
[5, 1, 1, 4, 10]
```

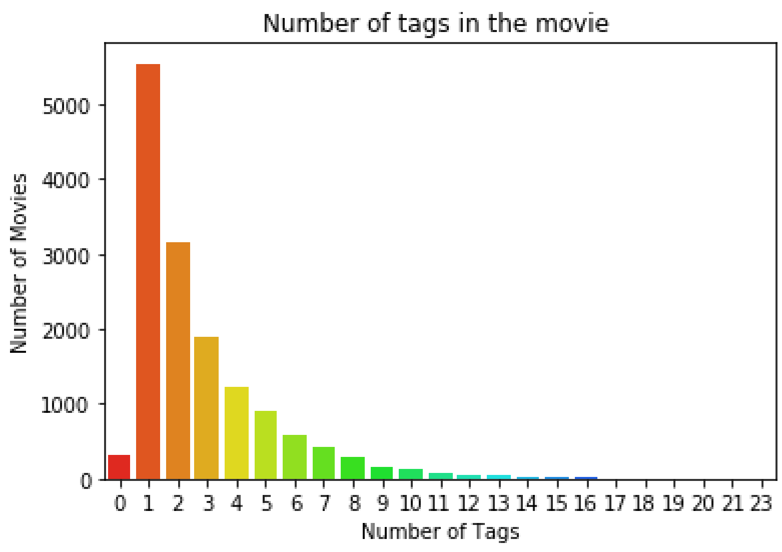
In [0]:

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

```
Maximum number of tags per movie: 23
Minimum number of tags per movie: 0
Avg. number of tags per movie: 2.794578
```

In [0]:

```
sns.countplot(tag_quest_count, palette='gist_rainbow')
plt.title("Number of tags in the movie ")
plt.xlabel("Number of Tags")
plt.ylabel("Number of Movies")
plt.show()
```



**Observations:**

1. **Maximum number of tags per movie: 25**
2. **Minimum number of tags per movie: 1**
3. **Avg. number of tags per movie: 2.981252**
4. **Most of the movie are having 2 or 3 tags**

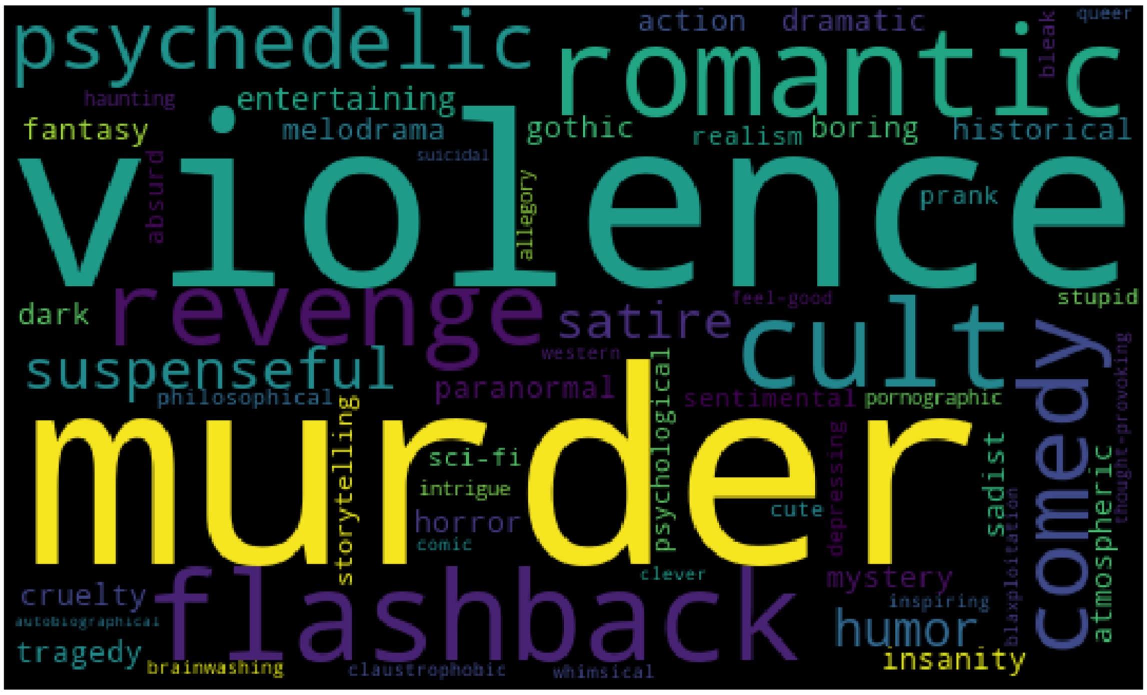
## 3.2.5 Most Frequent Tags

In [0]:

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

# Lets first convert the 'result' dictionary to 'list of tuples'
tup = dict(result.items())
#Initializing WordCloud using frequencies of tags.
wordcloud = WordCloud(background_color='black',
                          width=500,
                          height=300,
                    ).generate_from_frequencies(tup)

fig = plt.figure(figsize=(15,10))
plt.imshow(wordcloud)
plt.axis('off')
plt.tight_layout(pad=0)
fig.savefig("tag.png")
plt.show()
print("Time taken to run this cell :", datetime.now() - start)
```



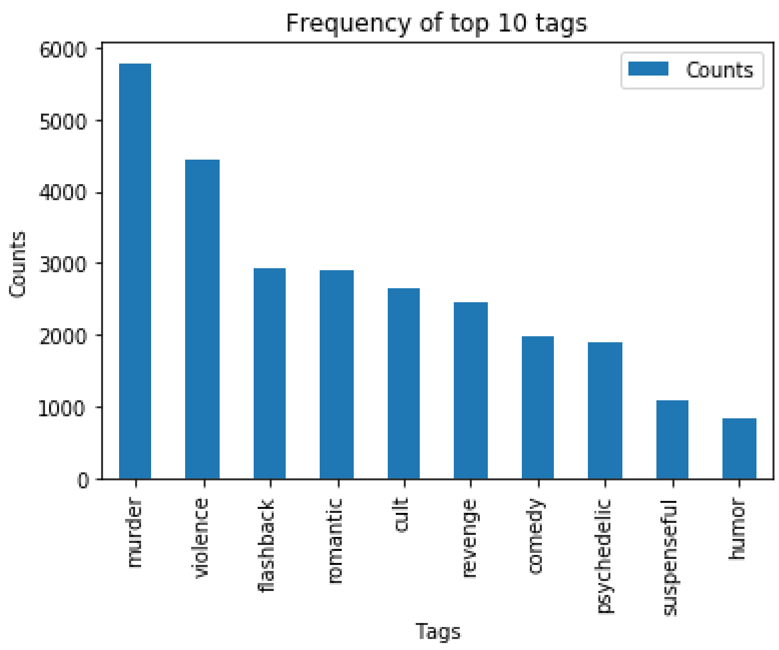
```
Time taken to run this cell : 0:00:00.607381
```

**Observations:**
**A look at the word cloud shows that "murder", "violence", "cult", "romantic", "flashback" are some of the most frequent tags.**

### 3.2.6 The top 10 tags

In [0]:

```
i=np.arange(10)
tag_df_sorted.head(10).plot(kind='bar')
plt.title('Frequency of top 10 tags')
plt.xticks(i, tag_df_sorted['Tags'])
plt.xlabel('Tags')
plt.ylabel('Counts')
plt.show()
```



**Observations:**

1. **Majority of the most frequent tags are one word common genre.**
2. **Murder is the top most frequent genre.**
3. **Violence, Flashback, romantic cult are some of the other famous tags.**

## 3.3 Cleaning and preprocessing of Plot_synopsis

### 3.3.1 Preprocessing

In [0]:

```
# https://gist.github.com/sebleier/554280
# we are removing the words from the stop words list: 'no', 'nor', 'not'
stopwords= ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you'r
e", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he', 'him',
'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 't
hey', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "th
at'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'ha
d', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as'
, 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through'
, 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'ov
er', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'how', 'all', 'an
y', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 'so', 'than', 'too'
, 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'no
w', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't",
'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "isn't", 'ma', 'migh
tn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'w
asn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"]
```

In [0]:

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

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

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

# remove all the unnecessay characters from the text
def removeSpecialChars(phrase):
    phrase = phrase.replace('\\r', ' ')
    phrase = phrase.replace('\\"', ' ')
    phrase = phrase.replace('\\n', ' ')
    phrase = re.sub('[^A-Za-z0-9]+', ' ', phrase)
    return phrase
```

In [23]:

```python
# Combining all the above stundents
from tqdm import tqdm
preprocessed_plot_synopsis = []
# tqdm is for printing the status bar
for sentance in tqdm(data['plot_synopsis'].values):
    sent = decontracted(sentance)
    sent = sent.replace('\\r', ' ')
    sent = sent.replace('\\"', ' ')
    sent = sent.replace('\\n', ' ')
    sent = re.sub('[^A-Za-z0-9]+', ' ', sent)
    # https://gist.github.com/sebleier/554280
    sent = ' '.join(e for e in sent.split() if e not in stopwords)
    preprocessed_plot_synopsis.append(sent.lower().strip())
```

```
100%|██████████| 14828/14828 [00:25<00:00, 575.39it/s]
```

In [24]:

```
print(preprocessed_plot_synopsis[100])
```

```
introductionmickey knox wife mallory go roadside caf new mexico desert the
y appear normal customers mickey eating key lime pie mallory dancing rock
n roll jukebox a group rednecks arrive one begins dancing flirting mallory
she encourages moment attacks without provocation smashing beer bottle dri
nks a fistfight ensues mallory beating man when redneck friend intervenes
mickey cuts one cowboy fingers stabs large bowie style knife mickey mallor
y begin killing diner patrons culminating morbid game eeny meeny miny moe
decide lives dies among last 2 customers after executing final victim mabe
l waitress couple make sure survivor cook remembers names promises tell au
thorities committed massacre embrace declare undying love part imickey mal
lory desert night mallory reminiscing first met a flashback shows mickey b
utcher deliveryman came house mallory lived abusive father rodney dangerfi
eld neglectful mother edie mcclurg younger brother kevin the flashback por
trayed 1950s type sitcom canned laughter track audience laughing hardest m
allory subjected lewd comments hints molestation father mickey instantly f
alls love mallory they leave together mickey stealing mallory father car m
ickey arrested imprisoned grand theft auto subsequently escapes horse work
farm tornado returns mallory house the two kill father beating jack handle
drowning fish tank burn mother alive bed they spare ten year old brother m
allory telling free they leave rapturous applause audience mickey mallory
get married side bridge exchanging snake wedding bands cutting palms mixin
g blood they drive motel night after watching television begin sex mickey
distracted female hostage furious mickey notion threesome mallory drives n
earby gas station flirts mechanic they begin sex hood car mallory angered
aggressive oral sex shoots death back motel mickey rapes hostage part iith
e pair continue killing spree bears similarities bonnie clyde starkweather
fugate murders ultimately claiming 48 victims along route 666 new mexico c
olorado utah pursuing two characters see murderers chance achieve fame the
first policeman detective jack scagnetti seems particularly fascinated mal
lory scagnetti already well known personality published author whose book
scagnetti scagnetti best seller within law enforcement scagnetti lifelong
obsession mass murderers witnessing mother shot killed charles whitman aus
tin texas eight despite heroic facade actually sociopathic strangling pros
titute death hotel room tries get rough scratches face the second follower
killers journalist wayne gale he australian hosts show called american man
iacs profiles mass murderers various clips mickey mallory shown gale actin
g outraged screen details pair crimes although air clearly regards crimes
fantastic way increasing show ratings it gale primarily responsible elevat
ing mickey mallory hero status show featuring interviews people around wor
ld expressing admiration killers mickey mallory become lost desert encount
er warren red cloud navajo indian pre adolescent grandson after two fall a
sleep navajo hoping expel demon perceives mickey begins chanting beside fi
re invoking nightmares mickey abusive parents mickey wakes rage fatally sh
oots navajo realizes it first time mallory mickey feel guilty murder while
fleeing scene desert stray field rattlesnakes bitten they drive drugstore
find snake antivenin pharmacist sets silent alarm mickey kills police cars
pull mallory captured immediately subsequently beaten sadistic brutish pol
ice a gunfight breaks mickey others scagnetti arrives he tells mickey unle
ss surrenders cut mallory breasts slashing times knife mickey gives gun at
tacks scagnetti knife the police use tazers scene ends mickey mallory beat
en group vengeful policemen part iiithe story picks one year later the hom
icidal couple imprisoned trial hinted footage provided gale both due moved
mental hospital declared insane scagnetti arrives prison encounters dwight
mcclusky sleazy abusive warden prison the two devise plan murder two crimi
nals mcclusky arrange scagnetti driver knox transfer alone pair scagnetti
murder claim tried escape wayne gale persuaded mickey agree live interview
air immediately super bowl mallory held solitary confinement elsewhere pri
son awaiting transport mental hospital during interview mickey gives speec
h murder provides enlightenment declares natural born killer his words ins
pire inmates watching interview tv recreation room incite riot mcclusky or
ders interview terminated gale violent protests mickey left alone gale gal
```

```
e tv crew several guards using lengthy joke diversion mickey overpowers gu
ard grabs shotgun kills guards gale crew mickey takes survivors hostage le
ading prison riot intending reunite mallory gale follows giving live telev
ision report people beaten killed around inmates torture murder prisoners 
guards alike during interview outbreak riot scagnetti mallory cell he atte
mpts seduce mallory plays along short time attacks scagnetti violently sma
shing face wall breaking nose the guards scagnetti subdue still live natio
nal television mickey breaks mallory cell engages brief mexican standoff s
cagnetti eventually feigning concession mallory approaches scagnetti behin
d slashes throat shank mickey reveals shotgun unloaded much scagnetti horr
or mallory picks scagnetti loaded gun kills they continue escape riot torn
prison two guards hostages the remainder gale tv crew killed gale snaps be
gins shoot guards claiming alive first time life after rescued mysterious 
prisoner named owen traft trio mickey mallory gale run warden mcclusky hea
vily armed posse guards cavanaugh guard would taken hostage shot death mcc
lusky men in retaliation mallory shoots gale thru hand retreat taking cove
r blood splattered shower room gale calls wife tells leaving he calls mist
ress tell see later however hangs mcclusky threatens storm shower room mic
key turn duct tapes shotguns necks gale guard homolka threatening kill liv
e tv the prisoners walk front door mcclusky guards massacred hordes inmate
s burst area they tear mcclusky apart cutting head displaying spike shot r
emoved theatrical showing seen director cut after escape owen never seen m
entioned mickey mallory steal van kill last guard escaping rural location 
give final interview wayne gale much surprise horror tell must also die ga
le attempts various arguments change minds finally appealing trademark pra
ctice leaving one witness tell tale mickey inform leaving camera witness g
ale accepts fate extends arms cross execute shooting numerous times unatte
nded camera continues roll the couple shown several years later rv mickey 
driving mallory pregt watching two children play
```

In [25]:

```
data.head()
```

Out[25]:

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

# 4. Machine Learning Models

## 4.1 Converting tags for multilabel problems

| X | y1 | y2 | y3 | y4 |
|---|---|---|---|---|
| x1 | 0 | 1 | 1 | 0 |
| x1 | 1 | 0 | 0 | 0 |
| x1 | 0 | 1 | 0 | 0 |

In [0]:

```
multilabel_vectorizer = CountVectorizer(binary='true', vocabulary=set_of_tags)
multilabel_yx = multilabel_vectorizer.fit_transform(preprocessed_tags)
```

In [27]:

```
multilabel_yx.shape
```

Out[27]:

```
(14828, 71)
```

In [28]:

```
print("Number of tags in sample :", multilabel_yx.shape[1])
```

```
Number of tags in sample : 71
```

## 4.2 Split the data into test and train

In [0]:

```
preprocessed_data = pd.DataFrame(preprocessed_plot_synopsis, columns=['plot_synopsis'])
```

In [0]:

```
total_size=preprocessed_data.shape[0]
train_size=int(0.80*total_size)

x_train=preprocessed_data.head(train_size)
x_test=preprocessed_data.tail(total_size - train_size)

y_train = multilabel_yx[0:train_size,:]
y_test = multilabel_yx[train_size:total_size,:]
```

In [0]:

```
print("Number of data points in train data X :", x_train.shape)
print("Number of data points in train data Y :", y_train.shape)

print("Number of data points in test data X :", x_test.shape)
print("Number of data points in test data Y :", y_test.shape)
```

```
Number of data points in train data X : (11862, 1)
Number of data points in train data Y : (11862, 71)
Number of data points in test data X : (2966, 1)
Number of data points in test data Y : (2966, 71)
```

## 4.3 Featurizing data : Hand-Crafted Features

In [0]:

```
fileSaveLocation=str(drivePath)+"MPST Dataset/data/"
```

### 4.3.1 Lexical :

#### 4.3.1.1 n-Gram : 1,2,3

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_word_n_gram.npz')) and (os.path.exist
s(str(fileSaveLocation)+'x_test_word_n_gram.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_word_n_gram = load_npz(str(fileSaveLocation)+'x_train_word_n_gram.npz')
  x_test_word_n_gram = load_npz(str(fileSaveLocation)+'x_test_word_n_gram.npz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  vectorizer = TfidfVectorizer(max_features=20000, ngram_range=(1,3),min_df=0.00009)
  x_train_word_n_gram = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_word_n_gram = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_word_n_gram.npz', x_train_word_n_gram)
  save_npz(str(fileSaveLocation)+'x_test_word_n_gram.npz', x_test_word_n_gram)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:01.863651
```

In [0]:

```
print(x_train_word_n_gram.shape)
print(x_test_word_n_gram.shape)
```

```
(11862, 20000)
(2966, 20000)
```

#### 4.3.1.2 Char n-Gram : 2,3

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_char_n_gram.npz')) and (os.path.exist
s(str(fileSaveLocation)+'x_test_char_n_gram.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_char_n_gram = load_npz(str(fileSaveLocation)+'x_train_char_n_gram.npz')
  x_test_char_n_gram = load_npz(str(fileSaveLocation)+'x_test_char_n_gram.npz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  vectorizer = TfidfVectorizer(max_features=20000, ngram_range=(2,3), analyzer='char_w
b', min_df=0.00009)
  x_train_char_n_gram = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_char_n_gram = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_char_n_gram.npz', x_train_char_n_gram)
  save_npz(str(fileSaveLocation)+'x_test_char_n_gram.npz', x_test_char_n_gram)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:03.343081
```

In [0]:

```
print(x_train_char_n_gram.shape)
print(x_test_char_n_gram.shape)
```

```
(11862, 13700)
(2966, 13700)
```

#### 4.3.1.3 - 2 Skip 2 Gram

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_2_skip_2_gram.npz')) and (os.path.exi
sts(str(fileSaveLocation)+'x_test_2_skip_2_gram.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_2_skip_2_gram = load_npz(str(fileSaveLocation)+'x_train_2_skip_2_gram.npz')
  x_test_2_skip_2_gram = load_npz(str(fileSaveLocation)+'x_test_2_skip_2_gram.npz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  skip2gram = skipgrams(x_train['plot_synopsis'], 2, 2)
  vectorizer = TfidfVectorizer(max_features=20000, analyzer='char_wb', vocabulary=skip2
gram, min_df=0.00009)
  x_train_2_skip_2_gram = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_2_skip_2_gram = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_2_skip_2_gram.npz', x_train_2_skip_2_gram)
  save_npz(str(fileSaveLocation)+'x_test_2_skip_2_gram.npz', x_test_2_skip_2_gram)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.685883
```

In [0]:

```
print(x_train_2_skip_2_gram.shape)
print(x_test_2_skip_2_gram.shape)
```

```
(11862, 35580)
(2966, 35580)
```

#### 4.3.1.4 - 2 Skip 3 Gram

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_2_skip_3_gram.npz')) and (os.path.exi
sts(str(fileSaveLocation)+'x_test_2_skip_3_gram.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_2_skip_3_gram = load_npz(str(fileSaveLocation)+'x_train_2_skip_3_gram.npz')
  x_test_2_skip_3_gram = load_npz(str(fileSaveLocation)+'x_test_2_skip_3_gram.npz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  skip3gram = skipgrams(x_train['plot_synopsis'], 3, 2)
  vectorizer = TfidfVectorizer(max_features=20000, analyzer='char_wb', vocabulary=skip3
gram, min_df=0.00009)
  x_train_2_skip_3_gram = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_2_skip_3_gram = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_2_skip_3_gram.npz', x_train_2_skip_3_gram)
  save_npz(str(fileSaveLocation)+'x_test_2_skip_3_gram.npz', x_test_2_skip_3_gram)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.660429
```

In [0]:

```
print(x_train_2_skip_3_gram.shape)
print(x_test_2_skip_3_gram.shape)
```

```
(11862, 71152)
(2966, 71152)
```

### 4.3.2 Bag Of Concepts : POS Tagging

In [0]:

```
# Load English tokenizer, tagger,
# parser, NER and word vectors
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("""My name is Shaurya Uppal.
I enjoy writing articles on GeeksforGeeks checkout
my other article by going to my profile section.""")

doc = nlp(text)

# Token and Tag
for token in doc:
  print(token, token.pos_)

# You want list of Verb tokens
print("Verbs:", [token.text for token in doc if token.pos_ == "VERB"])
```

```
My DET
name NOUN
is VERB
Shaurya PROPN
Uppal PROPN
. PUNCT

 SPACE
I PRON
enjoy VERB
writing VERB
articles NOUN
on ADP
GeeksforGeeks PROPN
checkout NOUN

 SPACE
my DET
other ADJ
article NOUN
by ADP
going VERB
to ADP
my DET
profile NOUN
section NOUN
. PUNCT
Verbs: ['is', 'enjoy', 'writing', 'going']
```

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_pos_tagging.csv')) and (os.path.exist
s(str(fileSaveLocation)+'x_test_pos_tagging.csv')):

  print("File Already Exist. Hence loading the data : ")
  x_train_pos_tagging = np.loadtxt(str(fileSaveLocation)+'x_train_pos_tagging.csv', del
imiter=',')
  x_test_pos_tagging = np.loadtxt(str(fileSaveLocation)+'x_test_pos_tagging.csv', delim
iter=',')

else:
  print("File Doesn't Exist. Hence Creating the data : ")

  vectorizer = CountVectorizer(max_features=20000, min_df=0.00009)
  x_train_plot_synopsis_BOW = vectorizer.fit_transform(x_train['plot_synopsis'])

  set_of_POS = set()

  for word in vectorizer.vocabulary_.keys():
    doc = nlp(word)
    for token in doc :
      #print(token.pos_)
      set_of_POS.add(token.pos_)

  print("Set of POS Available : ")
  print(set_of_POS)

  dict_of_POS = dict.fromkeys(set_of_POS, 0)

  x_train_pos_tagging = list()
  x_test_pos_tagging = list()

  dict_of_POS = dict.fromkeys(set_of_POS, 0)
  for sentence in x_train['plot_synopsis']:
      doc = nlp(sentence)
      dict_of_POS = dict.fromkeys(set_of_POS, 0)
      for token in doc :
        if token.pos_ in dict_of_POS.keys():
          dict_of_POS[token.pos_] = int(dict_of_POS[token.pos_])+1
      x_train_pos_tagging.append(list(dict_of_POS.values()))

  dict_of_POS = dict.fromkeys(set_of_POS, 0)
  for sentence in x_test['plot_synopsis']:
      doc = nlp(sentence)
      dict_of_POS = dict.fromkeys(set_of_POS, 0)
      for token in doc :
        if token.pos_ in dict_of_POS.keys():
          dict_of_POS[token.pos_] = int(dict_of_POS[token.pos_])+1
      x_test_pos_tagging.append(list(dict_of_POS.values()))

  x_train_pos_tagging = np.array(x_train_pos_tagging)
  x_test_pos_tagging = np.array(x_test_pos_tagging)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_pos_tagging.csv', x_train_pos_tagging, deli
miter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_pos_tagging.csv', x_test_pos_tagging, delimi
ter=',')
```

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.886901
```

In [0]:

```
print(x_train_pos_tagging.shape)
print(x_test_pos_tagging.shape)
```

```
(11862, 15)
(2966, 15)
```

### 4.3.3 Sentiments and Emotions :

In [0]:

```
sid = SentimentIntensityAnalyzer()

for_sentiment = 'a person is a person no matter how small dr seuss i teach the smallest
students with the biggest enthusiasm \
for learning my students learn in many different ways using all of our senses and multi
ple intelligences i use a wide range\
of techniques to help all my students succeed students in my class come from a variety
 of different backgrounds which makes\
for wonderful sharing of experiences and cultures including native americans our school
is a caring community of successful \
learners which can be seen through collaborative student project based learning in and
 out of the classroom kindergarteners \
in my class love to work with hands on materials and have many different opportunities
 to practice a skill before it is\
mastered having the social skills to work cooperatively with friends is a crucial aspec
t of the kindergarten curriculum\
montana is the perfect place to learn about agriculture and nutrition my students love
 to role play in our pretend kitchen\
in the early childhood classroom i have had several kids ask me can we try cooking with
real food i will take their idea \
and create common core cooking lessons where we learn important math and writing concep
ts while cooking delicious healthy \
food for snack time my students will have a grounded appreciation for the work that wen
t into making the food and knowledge \
of where the ingredients came from as well as how it is healthy for their bodies this p
roject would expand our learning of \
nutrition and agricultural cooking recipes by having us peel our own apples to make hom
emade applesauce make our own bread \
and mix up healthy plants from our classroom garden in the spring we will also create o
ur own cookbooks to be printed and \
shared with families students will gain math and literature skills as well as a life lo
ng enjoyment for healthy cooking \
nannan'
ss = sid.polarity_scores(for_sentiment)

for k in ss:
    print('{0}: {1}, '.format(k, ss[k]), end='')

# we can use these 4 things as features/attributes (neg, neu, pos, compound)
# neg: 0.0, neu: 0.753, pos: 0.247, compound: 0.93
```

```
neg: 0.01, neu: 0.745, pos: 0.245, compound: 0.9975,
```

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_sentiment_array.csv')) and (os.path.e
xists(str(fileSaveLocation)+'x_test_sentiment_array.csv')):

  print("File Already Exist. Hence loading the data : ")
  x_train_sentiment_array = np.loadtxt(str(fileSaveLocation)+'x_train_sentiment_array.c
sv', delimiter=',')
  x_test_sentiment_array = np.loadtxt(str(fileSaveLocation)+'x_test_sentiment_array.cs
v', delimiter=',')

else:
  print("File Doesn't Exist. Hence Creating the data : ")

  x_train_sentiment_array = list()
  x_test_sentiment_array = list()

  for sentence in x_train['plot_synopsis']:
    ss = sid.polarity_scores(sentence)
    x_train_sentiment_array.append([ss['neg'],ss['neu'],ss['pos'],ss['compound']])

  for sentence in x_test['plot_synopsis']:
    ss = sid.polarity_scores(sentence)
    x_test_sentiment_array.append([ss['neg'],ss['neu'],ss['pos'],ss['compound']])

  x_train_sentiment_array = np.array(x_train_sentiment_array)
  x_test_sentiment_array = np.array(x_test_sentiment_array)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_sentiment_array.csv', x_train_sentiment_arr
ay, delimiter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_sentiment_array.csv', x_test_sentiment_array
, delimiter=',')

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.752948
```

In [0]:

```
print(x_train_sentiment_array.shape)
print(x_test_sentiment_array.shape)
```

```
(11862, 4)
(2966, 4)
```

## 4.3.4 Word Embeddings :

#### 4.3.4.1 BOW

In [0]:

```
start = datetime.now()


if (os.path.exists(str(fileSaveLocation)+'x_train_word_uni_gram_bow.npz')) and (os.path
.exists(str(fileSaveLocation)+'x_test_word_uni_gram_bow.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_word_uni_gram_bow = load_npz(str(fileSaveLocation)+'x_train_word_uni_gram_bo
w.npz')
  x_test_word_uni_gram_bow = load_npz(str(fileSaveLocation)+'x_test_word_uni_gram_bow.n
pz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  vectorizer = CountVectorizer(max_features=20000, min_df=0.00009)
  x_train_word_uni_gram_bow = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_word_uni_gram_bow = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_word_uni_gram_bow.npz', x_train_word_uni_gram
_bow)
  save_npz(str(fileSaveLocation)+'x_test_word_uni_gram_bow.npz', x_test_word_uni_gram_b
ow)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:01.263558
```

In [0]:

```
print(x_train_word_uni_gram_bow.shape)
print(x_test_word_uni_gram_bow.shape)
```

```
(11862, 20000)
(2966, 20000)
```

### 4.3.4.2 TD-IDF

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_word_uni_gram_tfidf.npz')) and (os.pa
th.exists(str(fileSaveLocation)+'x_test_word_uni_gram_tfidf.npz')):
  print("File Already Exist. Hence loading the data : ")
  x_train_word_uni_gram_tfidf = load_npz(str(fileSaveLocation)+'x_train_word_uni_gram_t
fidf.npz')
  x_test_word_uni_gram_tfidf = load_npz(str(fileSaveLocation)+'x_test_word_uni_gram_tfi
df.npz')
else :
  print("File Doesn't Exist. Hence Creating the data : ")
  vectorizer = TfidfVectorizer(max_features=20000, min_df=0.00009)
  x_train_word_uni_gram_tfidf = vectorizer.fit_transform(x_train['plot_synopsis'])
  x_test_word_uni_gram_tfidf = vectorizer.transform(x_test['plot_synopsis'])

  print("Data Created. Saving the data for future reference : ")
  save_npz(str(fileSaveLocation)+'x_train_word_uni_gram_tfidf.npz', x_train_word_uni_gr
am_tfidf)
  save_npz(str(fileSaveLocation)+'x_test_word_uni_gram_tfidf.npz', x_test_word_uni_gram
_tfidf)


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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:01.404314
```

In [0]:

```
print(x_train_word_uni_gram_tfidf.shape)
print(x_test_word_uni_gram_tfidf.shape)
```

```
(11862, 20000)
(2966, 20000)
```

#### 4.3.4.3 Avg W2V

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_avg_w2v_vectors_plot_synopsis.csv'))
and (os.path.exists(str(fileSaveLocation)+'x_test_avg_w2v_vectors_plot_synopsis.csv')):

  print("File Already Exist. Hence loading the data : ")
  x_train_avg_w2v_vectors_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_train_avg
_w2v_vectors_plot_synopsis.csv', delimiter=',')
  x_test_avg_w2v_vectors_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_test_avg_w
2v_vectors_plot_synopsis.csv', delimiter=',')

else:
  print("File Doesn't Exist. Hence Creating the data : ")

  # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use
-pickle-to-save-and-load-variables-in-python/
  # make sure you have the glove_vectors file
  with open(str(drivePath)+'MPST Dataset/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())


  # Vectorizing Plot_synopsis for Train Data - AvgW2V
  x_train_avg_w2v_vectors_plot_synopsis = []; # the avg-w2v for each sentence is stored
in this list
  for sentence in x_train['plot_synopsis']: # for each sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_train_avg_w2v_vectors_plot_synopsis.append(vector)


  # Vectorizing Plot_synopsis for Test Data - AvgW2V
  x_test_avg_w2v_vectors_plot_synopsis = []; # the avg-w2v for each sentence is stored
 in this list
  for sentence in x_test['plot_synopsis']: # for each sentence
    vector = np.zeros(300) # as word vectors are of zero length
    cnt_words =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if word in glove_words:
            vector += model[word]
            cnt_words += 1
    if cnt_words != 0:
        vector /= cnt_words
    x_test_avg_w2v_vectors_plot_synopsis.append(vector)

  x_train_avg_w2v_vectors_plot_synopsis = np.array(x_train_avg_w2v_vectors_plot_synopsi
s)
  x_test_avg_w2v_vectors_plot_synopsis = np.array(x_test_avg_w2v_vectors_plot_synopsis)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_avg_w2v_vectors_plot_synopsis.csv', x_train
_avg_w2v_vectors_plot_synopsis, delimiter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_avg_w2v_vectors_plot_synopsis.csv', x_test_a
vg_w2v_vectors_plot_synopsis, delimiter=',')

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:04.562201
```

In [0]:

```
print(x_train_avg_w2v_vectors_plot_synopsis.shape)
print(x_test_avg_w2v_vectors_plot_synopsis.shape)
```

```
(11862, 300)
(2966, 300)
```

#### 4.3.4.3 TFIDF weighted AvgW2V

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_tfidf_w2v_vectors_plot_synopsis.csv'
)) and (os.path.exists(str(fileSaveLocation)+'x_test_tfidf_w2v_vectors_plot_synopsis.cs
v')):

  print("File Already Exist. Hence loading the data : ")
  x_train_tfidf_w2v_vectors_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_train_t
fidf_w2v_vectors_plot_synopsis.csv', delimiter=',')
  x_test_tfidf_w2v_vectors_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_test_tfi
df_w2v_vectors_plot_synopsis.csv', delimiter=',')

else:
  print("File Doesn't Exist. Hence Creating the data : ")

  # stronging variables into pickle files python: http://www.jessicayung.com/how-to-use
-pickle-to-save-and-load-variables-in-python/
  # make sure you have the glove_vectors file
  with open(str(drivePath)+'MPST Dataset/glove_vectors', 'rb') as f:
    model = pickle.load(f)
    glove_words =  set(model.keys())

  ## TFIDF weighted Avf W2V
  vectorizer = TfidfVectorizer(max_features=20000, min_df=0.00009)
  vectorizer.fit(x_train['plot_synopsis'])
  dictionary = dict(zip(vectorizer.get_feature_names(), list(vectorizer.idf_)))
  tfidf_words = set(vectorizer.get_feature_names())

  x_train_tfidf_w2v_vectors_plot_synopsis = []; # the avg-w2v for each sentence/review
 is stored in this list
  for sentence in x_train['plot_synopsis']: # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_train_tfidf_w2v_vectors_plot_synopsis.append(vector)


  # Vectorizing project title for Train Data - TFIDF weighted AvgW2V

  x_test_tfidf_w2v_vectors_plot_synopsis = []; # the avg-w2v for each sentence/review i
s stored in this list
  for sentence in x_test['plot_synopsis']: # for each review/sentence
    vector = np.zeros(300) # as word vectors are of zero length
    tf_idf_weight =0; # num of words with a valid vector in the sentence/review
    for word in sentence.split(): # for each word in a review/sentence
        if (word in glove_words) and (word in tfidf_words):
            vec = model[word] # getting the vector for each word
            # here we are multiplying idf value(dictionary[word]) and the tf value((sen
tence.count(word)/len(sentence.split())))
```

```
            tf_idf = dictionary[word]*(sentence.count(word)/len(sentence.split())) # ge
tting the tfidf value for each word
            vector += (vec * tf_idf) # calculating tfidf weighted w2v
            tf_idf_weight += tf_idf
    if tf_idf_weight != 0:
        vector /= tf_idf_weight
    x_test_tfidf_w2v_vectors_plot_synopsis.append(vector)

  x_train_tfidf_w2v_vectors_plot_synopsis = np.array(x_train_tfidf_w2v_vectors_plot_syn
opsis)
  x_test_tfidf_w2v_vectors_plot_synopsis = np.array(x_test_tfidf_w2v_vectors_plot_synop
sis)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_tfidf_w2v_vectors_plot_synopsis.csv', x_tra
in_tfidf_w2v_vectors_plot_synopsis, delimiter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_tfidf_w2v_vectors_plot_synopsis.csv', x_test
_tfidf_w2v_vectors_plot_synopsis, delimiter=',')

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:04.470896
```

In [0]:

```
print(x_train_tfidf_w2v_vectors_plot_synopsis.shape)
print(x_test_tfidf_w2v_vectors_plot_synopsis.shape)
```

```
(11862, 300)
(2966, 300)
```

## 4.3.5 Numerical Feature for Text :

### 4.3.5.1 Len of each Plot Synops

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_len_of_plot_synopsis.csv')) and (os.p
ath.exists(str(fileSaveLocation)+'x_test_len_of_plot_synopsis.csv')):

  print("File Already Exist. Hence loading the data : ")
  x_train_len_of_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_train_len_of_plot_
synopsis.csv', delimiter=',').reshape(-1,1)
  x_test_len_of_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_test_len_of_plot_sy
nopsis.csv', delimiter=',').reshape(-1,1)

else:
  x_train_len_of_plot_synopsis = list()
  x_test_len_of_plot_synopsis = list()
  for sentence in x_train['plot_synopsis']:
    x_train_len_of_plot_synopsis.append(len(sentence))

  for sentence in x_test['plot_synopsis']:
    x_test_len_of_plot_synopsis.append(len(sentence))

  x_train_len_of_plot_synopsis = np.array(x_train_len_of_plot_synopsis).reshape(-1,1)
  x_test_len_of_plot_synopsis = np.array(x_test_len_of_plot_synopsis).reshape(-1,1)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_len_of_plot_synopsis.csv', x_train_len_of_p
lot_synopsis, delimiter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_len_of_plot_synopsis.csv', x_test_len_of_plo
t_synopsis, delimiter=',')

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.831654
```

In [0]:

```
print(x_train_len_of_plot_synopsis.shape)
print(x_test_len_of_plot_synopsis.shape)
```

```
(11862, 1)
(2966, 1)
```

#### 4.3.5.2 Len of Unique words in Plot Synopsis

In [0]:

```
start = datetime.now()

if (os.path.exists(str(fileSaveLocation)+'x_train_len_unique_words_plot_synopsis.csv'))
and (os.path.exists(str(fileSaveLocation)+'x_test_len_unique_words_plot_synopsis.csv'
)):

  print("File Already Exist. Hence loading the data : ")
  x_train_len_unique_words_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_train_le
n_unique_words_plot_synopsis.csv', delimiter=',').reshape(-1,1)
  x_test_len_unique_words_plot_synopsis = np.loadtxt(str(fileSaveLocation)+'x_test_len_
unique_words_plot_synopsis.csv', delimiter=',').reshape(-1,1)

else:
  x_train_len_unique_words_plot_synopsis = list()
  x_test_len_unique_words_plot_synopsis = list()
  for sentence in x_train['plot_synopsis']:
    x_train_len_unique_words_plot_synopsis.append(len(Counter(sentence.split()).keys
()))

  for sentence in x_test['plot_synopsis']:
    x_test_len_unique_words_plot_synopsis.append(len(Counter(sentence.split()).keys()))

  x_train_len_unique_words_plot_synopsis = np.array(x_train_len_unique_words_plot_synop
sis).reshape(-1,1)
  x_test_len_unique_words_plot_synopsis = np.array(x_test_len_unique_words_plot_synopsi
s).reshape(-1,1)

  print("Data Created. Saving the data for future reference : ")
  np.savetxt(str(fileSaveLocation)+'x_train_len_unique_words_plot_synopsis.csv', x_trai
n_len_unique_words_plot_synopsis, delimiter=',')
  np.savetxt(str(fileSaveLocation)+'x_test_len_unique_words_plot_synopsis.csv', x_test_
len_unique_words_plot_synopsis, delimiter=',')

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

```
File Already Exist. Hence loading the data :
Time taken to run this cell : 0:00:00.757799
```

In [0]:

```
print(x_train_len_unique_words_plot_synopsis.shape)
print(x_test_len_unique_words_plot_synopsis.shape)
```

```
(11862, 1)
(2966, 1)
```

## 4.4 Combining All Features

In [0]:

```
print(x_train_word_n_gram.shape)
print(x_train_char_n_gram.shape)
print(x_train_2_skip_2_gram.shape)
print(x_train_2_skip_3_gram.shape)
print(x_train_pos_tagging.shape)
print(x_train_sentiment_array.shape)
print(x_train_word_uni_gram_bow.shape)
print(x_train_word_uni_gram_tfidf.shape)
print(x_train_avg_w2v_vectors_plot_synopsis.shape)
print(x_train_tfidf_w2v_vectors_plot_synopsis.shape)
print(x_train_len_of_plot_synopsis.shape)
print(x_train_len_unique_words_plot_synopsis.shape)
```

```
(11862, 20000)
(11862, 13700)
(11862, 35580)
(11862, 71152)
(11862, 15)
(11862, 4)
(11862, 20000)
(11862, 20000)
(11862, 300)
(11862, 300)
(11862, 1)
(11862, 1)
```

In [0]:

```
print(x_test_word_n_gram.shape)
print(x_test_char_n_gram.shape)
print(x_test_2_skip_2_gram.shape)
print(x_test_2_skip_3_gram.shape)
print(x_test_pos_tagging.shape)
print(x_test_sentiment_array.shape)
print(x_test_word_uni_gram_bow.shape)
print(x_test_word_uni_gram_tfidf.shape)
print(x_test_avg_w2v_vectors_plot_synopsis.shape)
print(x_test_tfidf_w2v_vectors_plot_synopsis.shape)
print(x_test_len_of_plot_synopsis.shape)
print(x_test_len_unique_words_plot_synopsis.shape)
```

```
(2966, 20000)
(2966, 13700)
(2966, 35580)
(2966, 71152)
(2966, 15)
(2966, 4)
(2966, 20000)
(2966, 20000)
(2966, 300)
(2966, 300)
(2966, 1)
(2966, 1)
```

In [0]:

```
x_train_features = hstack((x_train_word_n_gram ,x_train_char_n_gram ,x_train_2_skip_2_g
ram ,x_train_2_skip_3_gram ,x_train_pos_tagging ,x_train_sentiment_array ,x_train_word_
uni_gram_bow ,x_train_word_uni_gram_tfidf ,x_train_avg_w2v_vectors_plot_synopsis ,x_tra
in_tfidf_w2v_vectors_plot_synopsis ,x_train_len_of_plot_synopsis ,x_train_len_unique_wo
rds_plot_synopsis ))

x_test_features = hstack((x_test_word_n_gram ,x_test_char_n_gram ,x_test_2_skip_2_gram
,x_test_2_skip_3_gram ,x_test_pos_tagging ,x_test_sentiment_array ,x_test_word_uni_gram
_bow ,x_test_word_uni_gram_tfidf ,x_test_avg_w2v_vectors_plot_synopsis ,x_test_tfidf_w2
v_vectors_plot_synopsis ,x_test_len_of_plot_synopsis ,x_test_len_unique_words_plot_syno
psis ))
```

In [0]:

```
print(x_train_features.shape)
print(x_test_features.shape)
```

```
(11862, 181053)
(2966, 181053)
```

In [0]:

```
x_train_multilabel = x_train_features
x_test_multilabel = x_test_features
```

# 5. Models

## 5.1. Multiclass Classification : Machine Learning

### 5.1.1 One Vs Rest : Logistic Regression

In [0]:

```
start = datetime.now()

#https://datascience.stackexchange.com/questions/41680/how-to-implement-gridsearchcv-fo
r-onevsrestclassifier-of-logisticregression-clas
#when we want to apply gridsearch for oneVsRestClassifer then we need to give hyperpara
meter with prefix estimator__
alpha_params = [{'estimator__C':[0.00001, 0.0001, 0.001, 0.1, 1, 10, 100]}]
gridClassifer = OneVsRestClassifier(LogisticRegression(n_jobs=-1), n_jobs=-1)
GridSCv = GridSearchCV(gridClassifer, alpha_params, n_jobs=-1, return_train_score='Tru
e')
GridSCv.fit(x_train_multilabel,y_train)

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

```
Time taken to run this cell : 10:56:03.066483
```

In [0]:

```
print ("Best Estimator  : "+str(GridSCv.best_estimator_))
print ("Score of the model with CV  : " + str(GridSCv.score(x_train_multilabel,y_train
)))
print ("CV Results  : "+str(GridSCv.cv_results_))
```

```
Best Estimator  : OneVsRestClassifier(estimator=LogisticRegression(C=0.00
1, class_weight=None,
                                                 dual=False, fit_intercept
=True,
                                                 intercept_scaling=1,
                                                 l1_ratio=None, max_iter=1
00,
                                                 multi_class='warn', n_job
s=-1,
                                                 penalty='l2',
                                                 random_state=None,
                                                 solver='warn', tol=0.000
1,
                                                 verbose=0, warm_start=Fal
se),
                    n_jobs=-1)
Score of the model with CV  : 0.18774237059517787
CV Results  : {'mean_fit_time': array([ 616.88390191, 1153.63674053, 2695.
18325488, 4753.92774709,
       4919.10637514, 4878.22879052, 4639.14248141]), 'std_fit_time': arra
y([ 65.79240104, 117.48363966, 258.84812626, 542.4402409 ,
       265.44319435, 395.85397044, 271.93003244]), 'mean_score_time': arra
y([5.48470449, 5.12877409, 6.37301946, 5.41919327, 6.35972675,
       6.51857257, 4.43872595]), 'std_score_time': array([0.32636657, 0.39
864565, 1.05768989, 0.65494814, 0.50370566,
       0.81909382, 1.73763617]), 'param_estimator__C': masked_array(data=
[1e-05, 0.0001, 0.001, 0.1, 1, 10, 100],
             mask=[False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'estimator__C': 1e-05}, {'estimator
__C': 0.0001}, {'estimator__C': 0.001}, {'estimator__C': 0.1}, {'estimator
__C': 1}, {'estimator__C': 10}, {'estimator__C': 100}], 'split0_test_scor
e': array([0.01846232, 0.03009611, 0.0477997 , 0.04350025, 0.04324734,
       0.04223571, 0.04172989]), 'split1_test_score': array([0.02959029,
0.04982296, 0.07359636, 0.07814871, 0.07840162,
       0.07688417, 0.07764289]), 'split2_test_score': array([0.04653515,
0.06474456, 0.09610521, 0.08750632, 0.08270106,
       0.08851796, 0.08725341]), 'mean_test_score': array([0.03152925, 0.0
4822121, 0.07250042, 0.06971843, 0.06811668,
       0.06921261, 0.0688754 ]), 'std_test_score': array([0.01154241, 0.01
419045, 0.01973586, 0.01892856, 0.01767265,
       0.01965793, 0.01959166]), 'rank_test_score': array([7, 6, 1, 2, 5,
3, 4], dtype=int32), 'split0_train_score': array([0.03275164, 0.06208902,
0.18108245, 0.65655033, 0.63492666,
       0.63075367, 0.68158827]), 'split1_train_score': array([0.03174001,
0.06221548, 0.19600405, 0.77415276, 0.72192716,
       0.75986343, 0.73583713]), 'split2_train_score': array([0.03085483,
0.06057157, 0.20118867, 0.56385938, 0.55424886,
       0.61532625, 0.53705109]), 'mean_train_score': array([0.03178216, 0.
06162536, 0.19275839, 0.66485416, 0.63703423,
       0.66864778, 0.65149216]), 'std_train_score': array([0.00077494, 0.0
0074692, 0.00852314, 0.08605247, 0.0684706 ,
       0.06480598, 0.08389797])}
```

In [0]:

```
start = datetime.now()
classifier = OneVsRestClassifier(LogisticRegression(C=0.001, class_weight=None,
                                                    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)

classifier.fit(x_train_multilabel, y_train)
predictions_train = classifier.predict(x_train_multilabel)
predictions_test = classifier.predict(x_test_multilabel)
print("Time taken to run this cell :", datetime.now() - start)
```

In [0]:

```
print("Train Accuracy :",metrics.accuracy_score(y_train, predictions_train))
print("Train Hamming loss ",metrics.hamming_loss(y_train,predictions_train))

precision = precision_score(y_train, predictions_train, average='micro')
recall = recall_score(y_train, predictions_train, average='micro')
f1 = f1_score(y_train, predictions_train, average='micro')

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

precision = precision_score(y_train, predictions_train, average='macro')
recall = recall_score(y_train, predictions_train, average='macro')
f1 = f1_score(y_train, predictions_train, average='macro')

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

```
Train Accuracy : 0.18774237059517787
Train Hamming loss  0.028553719891427472
Train Micro-average quality numbers
Precision: 0.9166, Recall: 0.3399, F1-measure: 0.4959
Train Macro-average quality numbers
Precision: 0.7307, Recall: 0.1278, F1-measure: 0.2025
```

In [0]:

```
print("Test Accuracy :",metrics.accuracy_score(y_test, predictions_test))
print("Test Hamming loss ",metrics.hamming_loss(y_test,predictions_test))

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

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

precision = precision_score(y_test, predictions_test, average='macro')
recall = recall_score(y_test, predictions_test, average='macro')
f1 = f1_score(y_test, predictions_test, average='macro')

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

```
Test Accuracy : 0.09777478084962914
Test Hamming loss  0.027252523909471665
Test Micro-average quality numbers
Precision: 0.6024, Recall: 0.1581, F1-measure: 0.2505
Test Macro-average quality numbers
Precision: 0.1216, Recall: 0.0256, F1-measure: 0.0377
Time taken to run this cell : 0:42:44.263760
```

In [0]:

```
# predict probabilities
y_pred_prob_train = classifier.predict_proba(x_train_multilabel)

t = 0.25 # threshold value
y_pred_new_train = (y_pred_prob_train >= t).astype(int)

# evaluate performance
print("Train F1 Score with prob > 0.25 for each tags : ",f1_score(y_train, y_pred_new_t
rain, average="micro"))
```

```
Test F1 Score with prob > 0.25 for each tags :  0.6533833574983932
```

In [0]:

```
# predict probabilities
y_pred_prob_test = classifier.predict_proba(x_test_multilabel)

t = 0.25 # threshold value
y_pred_new_test = (y_pred_prob_test >= t).astype(int)

# evaluate performance
print("Test F1 Score with prob > 0.25 for each tags : ",f1_score(y_test, y_pred_new_tes
t, average="micro"))
```

### 5.1.2 One Vs Rest : MultinomialNB

In [0]:

```
start = datetime.now()

#https://datascience.stackexchange.com/questions/41680/how-to-implement-gridsearchcv-fo
r-onevsrestclassifier-of-logisticregression-clas
#when we want to apply gridsearch for oneVsRestClassifer then we need to give hyperpara
meter with prefix estimator__
alpha_params = [{'estimator__alpha':[0.00001, 0.0001, 0.001, 0.1, 1, 10, 100]}]
gridClassifer = OneVsRestClassifier(SGDClassifier(loss='hinge', n_jobs=-1), n_jobs=-1)
GridSCv = GridSearchCV(gridClassifer, alpha_params, n_jobs=-1, return_train_score='Tru
e')
GridSCv.fit(x_train_multilabel,y_train)

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

```
Time taken to run this cell : 0:32:28.209163
```

In [0]:

```
print ("Best Estimator  : "+str(GridSCv.best_estimator_))
print ("Score of the model with CV  : " + str(GridSCv.score(x_train_multilabel,y_train
)))
print ("CV Results  : "+str(GridSCv.cv_results_))
```

```
Best Estimator  : OneVsRestClassifier(estimator=SGDClassifier(alpha=100, a
verage=False,
                                            class_weight=None,
                                            early_stopping=False, epsilon=
0.1,
                                            eta0=0.0, fit_intercept=True,
                                            l1_ratio=0.15,
                                            learning_rate='optimal',
                                            loss='hinge', max_iter=1000,
                                            n_iter_no_change=5, n_jobs=Non
e,
                                            penalty='l2', power_t=0.5,
                                            random_state=None, shuffle=Tru
e,
                                            tol=0.001, validation_fraction
=0.1,
                                            verbose=0, warm_start=False),
                    n_jobs=-1)
Score of the model with CV  : 0.024026302478502782
CV Results  : {'mean_fit_time': array([348.54803173, 350.50825342, 346.608
60459, 345.68838247,
       325.69914921, 295.11560639, 207.2463905 ]), 'std_fit_time': array
([50.36549562, 50.41475082, 48.57534028, 55.88570026, 54.16011906,
       28.59028822, 20.26937631]), 'mean_score_time': array([11.63767171,
10.76464796, 10.3478748 , 11.55597305, 11.29857508,
        9.14647237,  7.82326849]), 'std_score_time': array([0.70847254, 0.
7975364 , 1.49123822, 1.47622698, 1.92281239,
       3.3298597 , 4.66607347]), 'param_estimator__alpha': masked_array(da
ta=[1e-05, 0.0001, 0.001, 0.1, 1, 10, 100],
             mask=[False, False, False, False, False, False, False],
       fill_value='?',
            dtype=object), 'params': [{'estimator__alpha': 1e-05}, {'estim
ator__alpha': 0.0001}, {'estimator__alpha': 0.001}, {'estimator__alpha':
0.1}, {'estimator__alpha': 1}, {'estimator__alpha': 10}, {'estimator__alph
a': 100}], 'split0_test_score': array([0.01896813, 0.00227618, 0.00050582,
0.        , 0.00758725,
       0.        , 0.01264542]), 'split1_test_score': array([0.0088518 ,
0.        , 0.        , 0.        , 0.00758725,
       0.05336368, 0.04400607]), 'split2_test_score': array([0.00075873,
0.01036925, 0.        , 0.00682853, 0.04881133,
       0.00252908, 0.03363682]), 'mean_test_score': array([0.00952622, 0.0
0421514, 0.00016861, 0.00227618, 0.02132861,
       0.01863092, 0.03009611]), 'std_test_score': array([0.00744924, 0.00
444972, 0.00023844, 0.003219  , 0.01943322,
       0.02458146, 0.01304544]), 'rank_test_score': array([4, 5, 7, 6, 2,
3, 1], dtype=int32), 'split0_train_score': array([0.04223571, 0.00265554,
0.00037936, 0.        , 0.01871523,
       0.        , 0.02921093]), 'split1_train_score': array([0.01011634,
0.        , 0.        , 0.        , 0.00872534,
       0.06259484, 0.04994942]), 'split2_train_score': array([0.00126454,
0.00859889, 0.        , 0.00316136, 0.02959029,
       0.00126454, 0.01770359]), 'mean_train_score': array([0.0178722 , 0.
00375148, 0.00012645, 0.00105379, 0.01901028,
       0.02128646, 0.03228798]), 'std_train_score': array([0.01760254, 0.0
03595  , 0.00017883, 0.00149028, 0.00852063,
       0.029214  , 0.0133429 ])}
```

In [0]:

```
start = datetime.now()
classifier = OneVsRestClassifier(SGDClassifier(alpha=100, average=False,
                                               class_weight=None,
                                               early_stopping=False, epsilon=0.1,
                                               eta0=0.0, fit_intercept=True,
                                               l1_ratio=0.15,
                                               learning_rate='optimal',
                                               loss='hinge', max_iter=1000,
                                               n_iter_no_change=5, n_jobs=-1,
                                               penalty='l2', power_t=0.5,
                                               random_state=None, shuffle=True,
                                               tol=0.001, validation_fraction=0.1,
                                               verbose=0, warm_start=False), n_jobs=-1)

classifier.fit(x_train_multilabel, y_train)

predictions_test = classifier.predict(x_test_multilabel)
print("Time taken to run this cell :", datetime.now() - start)
```

```
Time taken to run this cell : 0:02:08.850283
```

In [0]:

```
print("Test Accuracy :",metrics.accuracy_score(y_test, predictions_test))
print("Test Hamming loss ",metrics.hamming_loss(y_test,predictions_test))

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

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

precision = precision_score(y_test, predictions_test, average='macro')
recall = recall_score(y_test, predictions_test, average='macro')
f1 = f1_score(y_test, predictions_test, average='macro')

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

```
Test Accuracy : 0.032366824005394447
Test Hamming loss  0.028905055416789334
Test Micro-average quality numbers
Precision: 0.2500, Recall: 0.0018, F1-measure: 0.0036
Test Macro-average quality numbers
Precision: 0.0046, Recall: 0.0002, F1-measure: 0.0004
```

In [0]:

```
# predict probabilities
y_pred_prob_test = model.predict_proba(X_test)

t = 0.495 # threshold value
y_pred_new_test = (y_pred_prob_test >= t).astype(int)

# evaluate performance
print("Test F1 Score with prob > %f for each tags : %f" % (t,f1_score(Y_test, y_pred_ne
w_test, average="micro")))
```

```
Test F1 Score with prob > 0.495000 for each tags : 0.076542
```

## 5.2. Multiclass Classification : Deep Learning

In [0]:

```
# Load the TensorBoard notebook extension
%load_ext tensorboard.notebook
```

In [0]:

```
logdir = str(drivePath)+'MPST Dataset/logdir'

tensorboard_callback = TensorBoard(logdir, histogram_freq=1)
```

### 5.2.1 Defining Custom F1 Metrics Callbacks

In [0]:

```
class Metrics(Callback):
  def on_train_begin(self, logs={}):
    self.val_f1s = []
    self.val_recalls = []
    self.val_precisions = []

  def on_epoch_end(self, epoch, logs={}):
    val_predict = (np.asarray(self.model.predict(self.model.validation_data[0]))).round
()
    val_predict = np.where(val_predict > 0.32, 1, 0)
    val_targ = self.model.validation_data[1]
    _val_f1 = f1_score(val_targ, val_predict)
    _val_recall = recall_score(val_targ, val_predict)
    _val_precision = precision_score(val_targ, val_predict)
    self.val_f1s.append(_val_f1)
    self.val_recalls.append(_val_recall)
    self.val_precisions.append(_val_precision)
    print(" — val_f1: %f — val_precision: %f — val_recall %f" %(_val_f1, _val_precision
, _val_recall))
    return

metrics = Metrics()
```

### 5.2.2 Text Embedding

In [33]:

```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 50000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 300
# This is fixed.
EMBEDDING_DIM = 100
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@[\]^_`{|}~'
, lower=True)
tokenizer.fit_on_texts(preprocessed_data['plot_synopsis'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 124278 unique tokens.
```

In [34]:

```
X = tokenizer.texts_to_sequences(preprocessed_data['plot_synopsis'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (14828, 300)
```

In [35]:

```
Y = multilabel_yx
print('Shape of label tensor:', Y.shape)
```

```
Shape of label tensor: (14828, 71)
```

## 5.2.3 Splitting Train and Test Data (80:20)

In [36]:

```
X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.20, random_state
= 42)
print(X_train.shape,Y_train.shape)
print(X_test.shape,Y_test.shape)
```

```
(11862, 300) (11862, 71)
(2966, 300) (2966, 71)
```

## 5.2.1 Model 1 : Embedding + Conv1D + Conv1D + LSTM

In [0]:

```
model = Sequential()

model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(Dropout(0.5))
model.add(Conv1D(128, 3, padding='valid', activation='relu', strides=1,
                 kernel_initializer='he_normal',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))
model.add(Dropout(0.5))
model.add(Conv1D(86, 3, padding='valid', activation='relu', strides=1,
              kernel_initializer='he_normal',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))
model.add(LSTM(64, activation='relu', kernel_initializer='he_normal', recurrent_initial
izer='he_uniform',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))

model.add(Dense(Y.shape[1], activation='sigmoid'))
```

In [0]:

```
EPOCHS = 5
INIT_LR = 1e-2
BATCH_SIZE = 128
optimizer = Adadelta(lr=INIT_LR, clipnorm=1.0 , clipvalue=0.5)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_17"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_21 (Embedding)     (None, 300, 100)          5000000   
_________________________________________________________________
dropout_24 (Dropout)         (None, 300, 100)          0         
_________________________________________________________________
conv1d_36 (Conv1D)           (None, 298, 128)          38528     
_________________________________________________________________
dropout_25 (Dropout)         (None, 298, 128)          0         
_________________________________________________________________
conv1d_37 (Conv1D)           (None, 296, 86)           33110     
_________________________________________________________________
unified_lstm_30 (UnifiedLSTM (None, 64)                38656     
_________________________________________________________________
dense_20 (Dense)             (None, 71)                4615      
=================================================================
Total params: 5,114,909
Trainable params: 5,114,909
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
history = model.fit(X_train,
                    Y_train,
                    epochs=EPOCHS,
                    callbacks=[tensorboard_callback],
                    validation_data=(X_test,Y_test),
                    verbose=1,
                    use_multiprocessing=True,
                    batch_size=BATCH_SIZE
                    )
```

```
Train on 11862 samples, validate on 2966 samples
Epoch 1/5
11862/11862 [==============================] - 89s 7ms/sample - loss: 1243
9630886728180978155 52.0000 - accuracy: 0.5337 - val_loss: 19.4718 - val_ac
curacy: 0.5516
Epoch 2/5
11862/11862 [==============================] - 87s 7ms/sample - loss: 24.2
624 - accuracy: 0.5457 - val_loss: 17.0609 - val_accuracy: 0.6307
Epoch 3/5
11862/11862 [==============================] - 88s 7ms/sample - loss: 21.2
868 - accuracy: 0.5831 - val_loss: 15.2291 - val_accuracy: 0.7219
Epoch 4/5
11862/11862 [==============================] - 87s 7ms/sample - loss: 18.8
540 - accuracy: 0.6424 - val_loss: 13.8634 - val_accuracy: 0.8689
Epoch 5/5
11862/11862 [==============================] - 88s 7ms/sample - loss: 16.8
766 - accuracy: 0.7282 - val_loss: 12.8087 - val_accuracy: 0.9443
```

In [0]:

```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1]))
```

```
2966/2966 [==============================] - 20s 7ms/step
Test set
  Loss: 0.183
  Accuracy: 0.960
```

In [0]:

```
y_pred = model.predict(X_test)
```

In [0]:

```
index=178
```

In [0]:

```
y_pred[index]
```

Out[0]:

```
array([0.49761343, 0.49843895, 0.4962023 , 0.49814057, 0.4958822 ,
       0.49974406, 0.49666375, 0.49877727, 0.49790746, 0.49667192,
       0.49865994, 0.49831194, 0.4957823 , 0.49855024, 0.49669257,
       0.49589583, 0.49827236, 0.49750865, 0.49689147, 0.49716327,
       0.4978569 , 0.4989954 , 0.49725834, 0.4971361 , 0.49749827,
       0.49903372, 0.5004468 , 0.4963158 , 0.4986121 , 0.49870557,
       0.49890754, 0.49689347, 0.49675772, 0.49827784, 0.4970391 ,
       0.4977908 , 0.49728864, 0.49711162, 0.4994487 , 0.496768  ,
       0.4982046 , 0.49598023, 0.49794447, 0.49762693, 0.496953  ,
       0.49655133, 0.4979662 , 0.49653724, 0.4989409 , 0.49677646,
       0.4985376 , 0.49588796, 0.4994326 , 0.4976724 , 0.49755442,
       0.49642703, 0.49735466, 0.4961516 , 0.49834025, 0.49695057,
       0.5001777 , 0.49699318, 0.49746934, 0.49605468, 0.4973093 ,
       0.49741945, 0.49704978, 0.49529603, 0.4982547 , 0.4987413 ,
       0.49716786], dtype=float32)
```

In [0]:

```
Y_test[index].todense()
```

Out[0]:

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0]])
```

In [0]:

```
y_pred_prob = np.where(y_pred > 0.5, 1, 0)
```

In [0]:

```
y_pred_prob[index]
```

Out[0]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0])
```

In [0]:

```
multilabel_vectorizer.inverse_transform(Y_test[index].todense())
```

Out[0]:

```
[array(['cult', 'romantic', 'violence'], dtype='<U18')]
```

In [0]:

```
multilabel_vectorizer.inverse_transform(y_pred_prob[index])
```

Out[0]:

```
[array(['fantasy', 'sci-fi'], dtype='<U18')]
```

In [0]:

```
print("Test Accuracy :",accuracy_score(Y_test, y_pred_prob))
print("Test Hamming loss ",hamming_loss(Y_test,y_pred_prob))

precision = precision_score(Y_test, y_pred_prob, average='micro')
recall = recall_score(Y_test, y_pred_prob, average='micro')
f1 = f1_score(Y_test, y_pred_prob, average='micro')

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

precision = precision_score(Y_test, y_pred_prob, average='macro')
recall = recall_score(Y_test, y_pred_prob, average='macro')
f1 = f1_score(Y_test, y_pred_prob, average='macro')

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

```
Test Accuracy : 0.0064059339177343225
Test Hamming loss  0.055687462604351666
Test Micro-average quality numbers
Precision: 0.0303, Recall: 0.0129, F1-measure: 0.0181
Test Macro-average quality numbers
Precision: 0.0057, Recall: 0.0134, F1-measure: 0.0025
```

In [0]:

```
# predict probabilities
y_pred_prob_test = model.predict_proba(X_test)

t = 0.50 # threshold value
y_pred_new_test = (y_pred_prob_test >= t).astype(int)

# evaluate performance
print("Test F1 Score with prob > %f for each tags : %f" % (t,f1_score(Y_test, y_pred_ne
w_test, average="micro")))
```

```
Test F1 Score with prob > 0.500000 for each tags : 0.018086
```

In [0]:

```
# %tensorboard --logdir str(drivePath)+'MPST Dataset/logdir/'
%tensorboard --logdir '/content/drive/My Drive/Colab Notebooks/MPST Dataset/logdir/trai
n/'
```

### 5.2.2 Model 2 : Embedding + Conv1D + Conv1D + LSTM + LSTM

In [0]:

```
model = Sequential()

model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(Conv1D(128, 3, padding='valid', activation='relu', strides=1,
                 kernel_initializer='he_normal',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))
model.add(Dropout(0.5))
model.add(Conv1D(64, 3, padding='valid', activation='relu', strides=1,
              kernel_initializer='he_normal',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))

model.add(LSTM(32, kernel_initializer='he_normal', recurrent_initializer='he_uniform',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01), return_sequences=True))
model.add(BatchNormalization())
model.add(LSTM(8, activation='relu', kernel_initializer='he_normal', recurrent_initiali
zer='he_uniform',
              kernel_regularizer=regularizers.l2(0.01), activity_regularizer=regularize
rs.l1(0.01)))
model.add(Dense(Y.shape[1], activation='sigmoid'))
```

In [0]:

```
EPOCHS = 5
INIT_LR = 1e-2
BATCH_SIZE = 256
optimizer = RMSprop(lr=INIT_LR, decay=INIT_LR / 10 , clipnorm=1.0 , clipvalue=0.5)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_18"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_22 (Embedding)     (None, 300, 100)          5000000   
_________________________________________________________________
conv1d_38 (Conv1D)           (None, 298, 128)          38528     
_________________________________________________________________
dropout_26 (Dropout)         (None, 298, 128)          0         
_________________________________________________________________
conv1d_39 (Conv1D)           (None, 296, 64)           24640     
_________________________________________________________________
unified_lstm_31 (UnifiedLSTM (None, 296, 32)           12416     
_________________________________________________________________
batch_normalization_v2_15 (B (None, 296, 32)           128       
_________________________________________________________________
unified_lstm_32 (UnifiedLSTM (None, 8)                 1312      
_________________________________________________________________
dense_21 (Dense)             (None, 71)                639       
=================================================================
Total params: 5,077,663
Trainable params: 5,077,599
Non-trainable params: 64
_________________________________________________________________
```

In [0]:

```
history = model.fit(X_train,
                    Y_train,
                    epochs=EPOCHS,
                    #callbacks=[tensorboard_callback],
                    validation_data=(X_test,Y_test),
                    verbose=1,
                    use_multiprocessing=True,
                    batch_size=BATCH_SIZE
                    )
```

```
Train on 11862 samples, validate on 2966 samples
Epoch 1/5
11862/11862 [==============================] - 66s 6ms/sample - loss: 5.37
89 - accuracy: 0.9241 - val_loss: 0.4748 - val_accuracy: 0.9602
Epoch 2/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.44
34 - accuracy: 0.9614 - val_loss: 0.3966 - val_accuracy: 0.9602
Epoch 3/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.32
41 - accuracy: 0.9614 - val_loss: 0.2186 - val_accuracy: 0.9602
Epoch 4/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.25
01 - accuracy: 0.9614 - val_loss: 0.2879 - val_accuracy: 0.9602
Epoch 5/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.20
97 - accuracy: 0.9614 - val_loss: 0.1610 - val_accuracy: 0.9602
```

In [0]:

```
y_pred = model.predict(X_test)
```

In [0]:

```
index=250
```

In [0]:

```
y_pred[index]
```

Out[0]:

```
array([0.02176464, 0.0481205 , 0.00661826, 0.01341185, 0.00919336,
       0.00918382, 0.01009837, 0.03022772, 0.00799847, 0.00668123,
       0.00830293, 0.01824448, 0.03591272, 0.01297498, 0.00574744,
       0.00883302, 0.01079991, 0.13230374, 0.01218051, 0.03384197,
       0.17503479, 0.01688263, 0.03058138, 0.01780671, 0.03211951,
       0.05612874, 0.03969717, 0.00784889, 0.19307318, 0.00513455,
       0.03176627, 0.0088926 , 0.0142    , 0.03014848, 0.00697857,
       0.00717908, 0.03516746, 0.06064638, 0.04539773, 0.0131329 ,
       0.01558489, 0.00538221, 0.03399011, 0.3831901 , 0.03506595,
       0.00717473, 0.00514525, 0.03766787, 0.02000135, 0.00730088,
       0.01648283, 0.01970014, 0.12284148, 0.02267689, 0.01189485,
       0.02157739, 0.1649816 , 0.2117165 , 0.04691535, 0.05944139,
       0.00730658, 0.02051502, 0.02726656, 0.01758173, 0.00928724,
       0.07622096, 0.00810415, 0.04771796, 0.29041344, 0.00937271,
       0.0103147 ], dtype=float32)
```

In [0]:

```
Y_test[index].todense()
```

Out[0]:

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0]])
```

In [0]:

```
y_pred_prob = np.where(y_pred > 0.2, 1, 0)
```

In [0]:

```
y_pred_prob[index]
```

Out[0]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0])
```

In [0]:

```
multilabel_vectorizer.inverse_transform(Y_test[index].todense())
```

Out[0]:

```
[array(['flashback', 'murder'], dtype='<U18')]
```

In [0]:

```
multilabel_vectorizer.inverse_transform(y_pred_prob[index])
```

Out[0]:

```
[array(['murder', 'romantic', 'violence'], dtype='<U18')]
```

In [0]:

```
print("Test Accuracy :",accuracy_score(Y_test, y_pred_prob))
print("Test Hamming loss ",hamming_loss(Y_test,y_pred_prob))

precision = precision_score(Y_test, y_pred_prob, average='micro')
recall = recall_score(Y_test, y_pred_prob, average='micro')
f1 = f1_score(Y_test, y_pred_prob, average='micro')

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

precision = precision_score(Y_test, y_pred_prob, average='macro')
recall = recall_score(Y_test, y_pred_prob, average='macro')
f1 = f1_score(Y_test, y_pred_prob, average='macro')

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

```
Test Accuracy : 0.0023600809170600135
Test Hamming loss  0.05651847701176716
Test Micro-average quality numbers
Precision: 0.3021, Recall: 0.3208, F1-measure: 0.3111
Test Macro-average quality numbers
Precision: 0.0128, Recall: 0.0423, F1-measure: 0.0193
```

In [0]:

```
# predict probabilities
y_pred_prob_test = model.predict_proba(X_test)

t = 0.25 # threshold value
y_pred_new_test = (y_pred_prob_test >= t).astype(int)

# evaluate performance
print("Test F1 Score with prob > %f for each tags : %f" % (t,f1_score(Y_test, y_pred_ne
w_test, average="micro")))
```

```
Test F1 Score with prob > 0.250000 for each tags : 0.296534
```

In [0]:

```
%tensorboard --logdir '/content/drive/My Drive/Colab Notebooks/MPST Dataset/logdir/trai
n/'
```

```
Reusing TensorBoard on port 6009 (pid 11189), started 0:09:16 ago. (Use '!
kill 11189' to kill it.)
```

## 5.2.3 Model 3 : Embedding + Conv1D + BN + LSTM

In [0]:

```
X_train, X_test, y_train, y_test = train_test_split(preprocessed_data['plot_synopsis'].
values, multilabel_yx, test_size=0.20, random_state=42)
```

In [0]:

```
tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X_train)

X_train = tokenizer.texts_to_sequences(X_train)
X_test = tokenizer.texts_to_sequences(X_test)

vocab_size = len(tokenizer.word_index) + 1

maxlen = 200

X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)
```

#### Custom Embedding using Glove Vector

In [0]:

```
embeddings_dictionary = dict()

glove_file = open(str(drivePath)+'MPST Dataset/glove.6B.200d.txt', encoding="utf8")

for line in glove_file:
    records = line.split()
    word = records[0]
    vector_dimensions = asarray(records[1:], dtype='float32')
    embeddings_dictionary[word] = vector_dimensions
glove_file.close()

embedding_matrix = zeros((vocab_size, 200))
for word, index in tokenizer.word_index.items():
    embedding_vector = embeddings_dictionary.get(word)
    if embedding_vector is not None:
        embedding_matrix[index] = embedding_vector
```

In [0]:

```
deep_inputs = Input(shape=(maxlen,))
embedding_layer = Embedding(vocab_size, 200, weights=[embedding_matrix], trainable=True
)(deep_inputs)
LSTM_Layer_1 = LSTM(128)(embedding_layer)
batch_normal = BatchNormalization()(LSTM_Layer_1)
dense_layer_1 = Dense(y_train.shape[1], activation='sigmoid')(batch_normal)
model = Model(inputs=deep_inputs, outputs=dense_layer_1)
```

In [41]:

```
EPOCHS = 5
INIT_LR = 1e-3
BATCH_SIZE = 128
optimizer = RMSprop(lr=INIT_LR, decay=INIT_LR / 10, clipnorm=1.0, clipvalue=0.5)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_1 (InputLayer)         [(None, 200)]             0         
_________________________________________________________________
embedding (Embedding)        (None, 200, 200)          22375200  
_________________________________________________________________
unified_lstm (UnifiedLSTM)   (None, 128)               168448    
_________________________________________________________________
batch_normalization_v2 (Batc (None, 128)               512       
_________________________________________________________________
dense (Dense)                (None, 71)                9159      
=================================================================
Total params: 22,553,319
Trainable params: 22,553,063
Non-trainable params: 256
_________________________________________________________________
```

In [0]:

```
history = model.fit(X_train,
                    y_train,
                    epochs=EPOCHS,
                    callbacks=[tensorboard_callback],
                    validation_data=(X_test,Y_test),
                    verbose=1,
                    use_multiprocessing=True,
                    batch_size=BATCH_SIZE
                    )
```

```
Train on 11862 samples, validate on 2966 samples
Epoch 1/5
11862/11862 [==============================] - 65s 5ms/sample - loss: 0.48
82 - accuracy: 0.8554 - val_loss: 0.2496 - val_accuracy: 0.9602
Epoch 2/5
11862/11862 [==============================] - 65s 5ms/sample - loss: 0.15
57 - accuracy: 0.9632 - val_loss: 0.1296 - val_accuracy: 0.9602
Epoch 3/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.10
52 - accuracy: 0.9657 - val_loss: 0.1263 - val_accuracy: 0.9602
Epoch 4/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.09
29 - accuracy: 0.9684 - val_loss: 0.1281 - val_accuracy: 0.9603
Epoch 5/5
11862/11862 [==============================] - 64s 5ms/sample - loss: 0.08
30 - accuracy: 0.9713 - val_loss: 0.1285 - val_accuracy: 0.9611
```

In [0]:

```
y_pred = model.predict(X_test)
```

In [0]:

```
index=80
```

In [0]:

```
y_pred[index]
```

Out[0]:

```
array([3.54459882e-03, 3.41361463e-02, 2.41994858e-05, 3.97709012e-03,
       6.10053539e-05, 4.55975533e-05, 4.47928905e-05, 1.23710632e-02,
       1.75496936e-03, 4.49419022e-05, 3.32587957e-03, 5.89433312e-03,
       1.31726265e-02, 5.72419167e-03, 3.88920307e-05, 6.39677048e-03,
       4.37545776e-03, 4.02636230e-02, 4.11307812e-03, 1.04025900e-02,
       4.42006886e-02, 2.58091092e-03, 1.54050291e-02, 5.99548221e-03,
       3.16424370e-02, 1.40354931e-02, 5.10680676e-03, 3.74019146e-05,
       1.72877163e-01, 6.59525394e-05, 1.06664300e-02, 4.90248203e-05,
       4.39560413e-03, 9.70086455e-03, 3.64482403e-05, 3.01003456e-05,
       6.37644529e-03, 2.01436877e-02, 1.29881203e-02, 5.22288680e-03,
       1.03268623e-02, 3.07559967e-05, 7.26589561e-03, 6.43350363e-01,
       5.07995188e-02, 6.29723072e-05, 5.73396683e-05, 2.30746567e-02,
       6.34410977e-03, 4.38094139e-05, 9.62069631e-03, 5.27071953e-03,
       2.51215994e-02, 1.04426146e-02, 2.03591585e-03, 1.89269483e-02,
       2.83967733e-01, 5.63430786e-02, 4.69703078e-02, 5.05310595e-02,
       4.75943089e-05, 2.04059482e-03, 1.27407312e-02, 8.29064846e-03,
       4.30798531e-03, 1.66011482e-01, 5.39720058e-05, 5.88383079e-02,
       3.92955571e-01, 4.70462441e-03, 8.94844532e-04], dtype=float32)
```

In [0]:

```
Y_test[index].todense()
```

Out[0]:

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0]])
```

In [0]:

```
y_pred_prob = np.where(y_pred > 0.1, 1, 0)
```

In [0]:

```
y_pred_prob[index]
```

Out[0]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 1, 0, 0])
```

In [0]:

```
multilabel_vectorizer.inverse_transform(Y_test[index].todense())
```

Out[0]:

```
[array(['claustrophobic', 'flashback', 'murder', 'violence'], dtype='<U1
8')]
```

In [0]:

```
multilabel_vectorizer.inverse_transform(y_pred_prob[index])
```

Out[0]:

```
[array(['flashback', 'murder', 'revenge', 'suspenseful', 'violence'],
       dtype='<U18')]
```

In [0]:

```
print("Test Accuracy :",accuracy_score(Y_test, y_pred_prob))
print("Test Hamming loss ",hamming_loss(Y_test,y_pred_prob))

precision = precision_score(Y_test, y_pred_prob, average='micro')
recall = recall_score(Y_test, y_pred_prob, average='micro')
f1 = f1_score(Y_test, y_pred_prob, average='micro')

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

precision = precision_score(Y_test, y_pred_prob, average='macro')
recall = recall_score(Y_test, y_pred_prob, average='macro')
f1 = f1_score(Y_test, y_pred_prob, average='macro')

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

```
Test Accuracy : 0.004045853000674309
Test Hamming loss  0.07193735575964214
Test Micro-average quality numbers
Precision: 0.2722, Recall: 0.4827, F1-measure: 0.3481
Test Macro-average quality numbers
Precision: 0.0966, Recall: 0.0839, F1-measure: 0.0496
```

### 5.2.4 Model 4 : Embedding + Conv1D + Conv1D + LSTM

In [0]:

```
deep_inputs = Input(shape=(maxlen,))

embedding_layer = Embedding(vocab_size, 200, weights=[embedding_matrix], trainable=True
)(deep_inputs)

conv1d_layer1 = Conv1D(300, 3, padding='valid', activation='relu', strides=1, kernel_in
itializer='he_normal',
              kernel_regularizer=regularizers.l2(0.0000001), activity_regularizer=regul
arizers.l1(0.0000001))(embedding_layer)

dropout_1 = Dropout(0.5)(conv1d_layer1)

conv1d_layer2 = Conv1D(150, 3, padding='valid', activation='relu', strides=1, kernel_in
itializer='he_normal',
              kernel_regularizer=regularizers.l2(0.0000001), activity_regularizer=regul
arizers.l1(0.0000001))(dropout_1)

LSTM_Layer_1 = LSTM(128, activation='relu', kernel_initializer='he_normal', recurrent_i
nitializer='he_uniform',
              kernel_regularizer=regularizers.l2(0.0000001), activity_regularizer=regul
arizers.l1(0.0000001))(conv1d_layer2)

dense_layer_1 = Dense(y_train.shape[1], activation='sigmoid')(LSTM_Layer_1)

model = Model(inputs=deep_inputs, outputs=dense_layer_1)
```

In [0]:

```
EPOCHS = 5
INIT_LR = 1e-3
BATCH_SIZE = 128
optimizer = Adadelta(lr=INIT_LR, clipnorm=1.0, clipvalue=0.5)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "model_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
input_4 (InputLayer)         [(None, 200)]             0         
_________________________________________________________________
embedding_8 (Embedding)      (None, 200, 200)          22375200  
_________________________________________________________________
conv1d_12 (Conv1D)           (None, 198, 300)          180300    
_________________________________________________________________
dropout_9 (Dropout)          (None, 198, 300)          0         
_________________________________________________________________
conv1d_13 (Conv1D)           (None, 196, 150)          135150    
_________________________________________________________________
unified_lstm_13 (UnifiedLSTM (None, 128)               142848    
_________________________________________________________________
dense_8 (Dense)              (None, 71)                9159      
=================================================================
Total params: 22,842,657
Trainable params: 22,842,657
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
history = model.fit(X_train,
                    y_train,
                    epochs=EPOCHS,
                    callbacks=[tensorboard_callback],
                    validation_data=(X_test,Y_test),
                    verbose=1,
                    use_multiprocessing=True,
                    batch_size=BATCH_SIZE
                    )
```

```
Train on 11862 samples, validate on 2966 samples
Epoch 1/5
11862/11862 [==============================] - 137s 12ms/sample - loss: 0.
6985 - accuracy: 0.4804 - val_loss: 0.6951 - val_accuracy: 0.4972
Epoch 2/5
11862/11862 [==============================] - 136s 11ms/sample - loss: 0.
6964 - accuracy: 0.4954 - val_loss: 0.6943 - val_accuracy: 0.5036
Epoch 3/5
11862/11862 [==============================] - 136s 11ms/sample - loss: 0.
6950 - accuracy: 0.5045 - val_loss: 0.6935 - val_accuracy: 0.5065
Epoch 4/5
11862/11862 [==============================] - 136s 11ms/sample - loss: 0.
6939 - accuracy: 0.5102 - val_loss: 0.6928 - val_accuracy: 0.5064
Epoch 5/5
11862/11862 [==============================] - 136s 11ms/sample - loss: 0.
6929 - accuracy: 0.5120 - val_loss: 0.6921 - val_accuracy: 0.5121
```

In [0]:

```
y_pred = model.predict(X_test)
```

In [0]:

```
index=100
```

In [0]:

```
y_pred[index]
```

Out[0]:

```
array([0.48572654, 0.4872131 , 0.5260199 , 0.49743268, 0.45264584,
       0.48628697, 0.5307601 , 0.53111356, 0.553022  , 0.49536976,
       0.51513445, 0.5090978 , 0.49438146, 0.5325114 , 0.544233  ,
       0.47076324, 0.48784718, 0.5163332 , 0.46566328, 0.52209234,
       0.46706054, 0.51949745, 0.46928328, 0.46751514, 0.48264486,
       0.5098025 , 0.4603574 , 0.5092606 , 0.5079524 , 0.512665  ,
       0.5068102 , 0.5186752 , 0.4978585 , 0.47777888, 0.5282985 ,
       0.5237581 , 0.47770587, 0.46726334, 0.49336836, 0.5166716 ,
       0.47496453, 0.4825063 , 0.49750024, 0.4938319 , 0.51429695,
       0.45451275, 0.51228976, 0.54052   , 0.50562316, 0.49801996,
       0.47444528, 0.48635966, 0.50795114, 0.4748338 , 0.49134123,
       0.52502257, 0.5201552 , 0.5336242 , 0.48640525, 0.50362694,
       0.5093018 , 0.54585385, 0.50559306, 0.5181275 , 0.46874648,
       0.5138421 , 0.44396222, 0.4327832 , 0.514077  , 0.503973  ,
       0.4810241 ], dtype=float32)
```

In [0]:

```
Y_test[index].todense()
```

Out[0]:

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0]])
```

In [0]:

```
y_pred_prob = np.where(y_pred > 0.54, 1, 0)
```

In [0]:

```
y_pred_prob[index]
```

Out[0]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
       0, 0, 0, 0, 0])
```

In [0]:

```
multilabel_vectorizer.inverse_transform(Y_test[index].todense())
```

Out[0]:

```
[array(['fantasy', 'satire'], dtype='<U18')]
```

In [0]:

```
multilabel_vectorizer.inverse_transform(y_pred_prob[index])
```

Out[0]:

```
[array(['autobiographical', 'christian film', 'paranormal', 'sentimenta
l'],
       dtype='<U18')]
```

In [0]:

```
print("Test Accuracy :",accuracy_score(Y_test, y_pred_prob))
print("Test Hamming loss ",hamming_loss(Y_test,y_pred_prob))

precision = precision_score(Y_test, y_pred_prob, average='micro')
recall = recall_score(Y_test, y_pred_prob, average='micro')
f1 = f1_score(Y_test, y_pred_prob, average='micro')

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

precision = precision_score(Y_test, y_pred_prob, average='macro')
recall = recall_score(Y_test, y_pred_prob, average='macro')
f1 = f1_score(Y_test, y_pred_prob, average='macro')

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

```
Test Accuracy : 0.006068779501011463
Test Hamming loss  0.06295290285204144
Test Micro-average quality numbers
Precision: 0.0087, Recall: 0.0051, F1-measure: 0.0064
Test Macro-average quality numbers
Precision: 0.0006, Recall: 0.0202, F1-measure: 0.0011
```

### 5.2.5 Model 5 : Embedding + Conv1D + BN + Conv1D + BN + LSTM

In [0]:

```
model = Sequential()

model.add(Embedding(vocab_size, 200, weights=[embedding_matrix], trainable=False, input
_length=X_train.shape[1]))

model.add(Conv1D(100, 3, padding='valid', strides=1,
                 kernel_initializer='glorot_normal', kernel_regularizer=regularizers.l2
(0.0001), activity_regularizer=regularizers.l1(0.0001)))

model.add(Dropout(0.3))

model.add(Conv1D(50, 3, padding='valid', activation='relu', strides=1,
              kernel_initializer='he_uniform', kernel_regularizer=regularizers.l2(0.000
1), activity_regularizer=regularizers.l1(0.0001)))

model.add(BatchNormalization())

model.add(LSTM(64, activation='relu', kernel_initializer='he_normal',
              kernel_regularizer=regularizers.l2(0.0001), activity_regularizer=regulari
zers.l1(0.0001)))

model.add(Dense(Y.shape[1], activation='sigmoid'))
```

In [0]:

```
EPOCHS = 10
INIT_LR = 0.0001
BATCH_SIZE = 128
optimizer = Adam(lr=INIT_LR, decay=INIT_LR / 10 , clipnorm=1.0, clipvalue=0.5)
model.compile(optimizer=optimizer, loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
```

```
Model: "sequential_8"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_12 (Embedding)     (None, 200, 200)          22375200  
_________________________________________________________________
conv1d_18 (Conv1D)           (None, 198, 100)          60100     
_________________________________________________________________
dropout_12 (Dropout)         (None, 198, 100)          0         
_________________________________________________________________
conv1d_19 (Conv1D)           (None, 196, 50)           15050     
_________________________________________________________________
batch_normalization_v2_9 (Ba (None, 196, 50)           200       
_________________________________________________________________
unified_lstm_16 (UnifiedLSTM (None, 64)                29440     
_________________________________________________________________
dense_11 (Dense)             (None, 71)                4615      
=================================================================
Total params: 22,484,605
Trainable params: 109,305
Non-trainable params: 22,375,300
_________________________________________________________________
```

In [0]:

```
history = model.fit(X_train,
                    y_train,
                    epochs=EPOCHS,
                    callbacks=[tensorboard_callback],
                    validation_data=(X_test,Y_test),
                    verbose=1,
                    use_multiprocessing=True,
                    batch_size=BATCH_SIZE
                    )
```

```
Train on 11862 samples, validate on 2966 samples
Epoch 1/10
11862/11862 [==============================] - 124s 10ms/sample - loss: 0.
5575 - accuracy: 0.9547 - val_loss: 0.4930 - val_accuracy: 0.9535
Epoch 2/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
4677 - accuracy: 0.9575 - val_loss: 0.4488 - val_accuracy: 0.9579
Epoch 3/10
11862/11862 [==============================] - 121s 10ms/sample - loss: 0.
4092 - accuracy: 0.9597 - val_loss: 0.4131 - val_accuracy: 0.9596
Epoch 4/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3798 - accuracy: 0.9605 - val_loss: 0.3795 - val_accuracy: 0.9600
Epoch 5/10
11862/11862 [==============================] - 121s 10ms/sample - loss: 0.
3598 - accuracy: 0.9608 - val_loss: 0.3638 - val_accuracy: 0.9600
Epoch 6/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3449 - accuracy: 0.9609 - val_loss: 0.3463 - val_accuracy: 0.9601
Epoch 7/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3330 - accuracy: 0.9611 - val_loss: 0.3329 - val_accuracy: 0.9601
Epoch 8/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3230 - accuracy: 0.9612 - val_loss: 0.3213 - val_accuracy: 0.9601
Epoch 9/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3144 - accuracy: 0.9612 - val_loss: 0.3132 - val_accuracy: 0.9602
Epoch 10/10
11862/11862 [==============================] - 120s 10ms/sample - loss: 0.
3067 - accuracy: 0.9613 - val_loss: 0.3046 - val_accuracy: 0.9602
```

In [0]:

```
y_pred = model.predict(X_test)
```

In [0]:

```
index=100
```

In [0]:

```
y_pred[index]
```

Out[0]:

```
array([0.00757533, 0.03392974, 0.00059402, 0.00835755, 0.0023534 ,
       0.00316611, 0.00050309, 0.00959319, 0.00122949, 0.00098664,
       0.00344941, 0.0076693 , 0.01407292, 0.00248823, 0.0017181 ,
       0.00143152, 0.00362474, 0.11732981, 0.00269163, 0.01539421,
       0.12749282, 0.00463751, 0.01090479, 0.00776958, 0.0205375 ,
       0.0270347 , 0.02132046, 0.00050887, 0.1591017 , 0.00329709,
       0.01893055, 0.00112686, 0.00236776, 0.01038629, 0.00042719,
       0.00094864, 0.01458412, 0.05390108, 0.02462414, 0.0022983 ,
       0.00541914, 0.00190628, 0.01464322, 0.36199924, 0.01832333,
       0.00150514, 0.00150383, 0.01843822, 0.00592199, 0.00322202,
       0.00353482, 0.01091373, 0.1121878 , 0.00761327, 0.00608477,
       0.0094474 , 0.12635192, 0.174972  , 0.02378079, 0.03229901,
       0.00158066, 0.00895789, 0.01346084, 0.00718784, 0.00214452,
       0.04474732, 0.00751063, 0.01884264, 0.2663786 , 0.00135651,
       0.00267184], dtype=float32)
```

In [0]:

```
Y_test[index].todense()
```

Out[0]:

```
matrix([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0, 0, 0]])
```

In [0]:

```
y_pred_prob = np.where(y_pred > 0.15, 1, 0)
```

In [0]:

```
y_pred_prob[index]
```

Out[0]:

```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 0])
```

In [0]:

```
multilabel_vectorizer.inverse_transform(Y_test[index].todense())
```

Out[0]:

```
[array(['fantasy', 'satire'], dtype='<U18')]
```

In [0]:

```
multilabel_vectorizer.inverse_transform(y_pred_prob[index])
```

Out[0]:

```
[array(['flashback', 'murder', 'romantic', 'violence'], dtype='<U18')]
```

In [0]:

```
print("Test Accuracy :",accuracy_score(Y_test, y_pred_prob))
print("Test Hamming loss ",hamming_loss(Y_test,y_pred_prob))

precision = precision_score(Y_test, y_pred_prob, average='micro')
recall = recall_score(Y_test, y_pred_prob, average='micro')
f1 = f1_score(Y_test, y_pred_prob, average='micro')

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

precision = precision_score(Y_test, y_pred_prob, average='macro')
recall = recall_score(Y_test, y_pred_prob, average='macro')
f1 = f1_score(Y_test, y_pred_prob, average='macro')

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

```
Test Accuracy : 0.0006743088334457181
Test Hamming loss  0.07470107224601825
Test Micro-average quality numbers
Precision: 0.2566, Recall: 0.4625, F1-measure: 0.3301
Test Macro-average quality numbers
Precision: 0.0346, Recall: 0.0736, F1-measure: 0.0370
```

In [0]:

```
# predict probabilities
y_pred_prob_test = model.predict_proba(X_test)

t = 0.15 # threshold value
y_pred_new_test = (y_pred_prob_test >= t).astype(int)

# evaluate performance
print("Test F1 Score with prob > %f for each tags : %f" % (t,f1_score(Y_test, y_pred_new_test, average="micro")))
```

```
Test F1 Score with prob > 0.150000 for each tags : 0.330111
```

## Models Summary :

### Machine Learning :

In [0]:

```
x = PrettyTable()

x.field_names = ["Model", "Precision", "Recall", "Micro : F1-Score"]

x.add_row(["Logistic Regression", 0.6024, 0.1581, 0.4015])
x.add_row(["SVM", 0.2500, 0.0018, 0.0036])

print(x)
```

```
+---------------------+-----------+--------+------------------+
|        Model        | Precision | Recall | Micro : F1-Score |
+---------------------+-----------+--------+------------------+
| Logistic Regression |   0.6024  | 0.1581 |      0.4015      |
|         SVM         |    0.25   | 0.0018 |      0.0036      |
+---------------------+-----------+--------+------------------+
```

## Deep Learning :

In [42]:

```
x = PrettyTable()

x.field_names = ["Model #", "Precision", "Recall", "Micro : F1-Score"]

x.add_row(["Model 1",0.0303,0.0129,0.0181])
x.add_row(["Model 2",0.3021,0.3208,0.3111])
x.add_row(["Model 3",0.2722,0.4827,0.3481])
x.add_row(["Model 4",0.0087,0.0051,0.0064])
x.add_row(["Model 5",0.2566,0.4625,0.3301])

print(x)
```

```
+---------+-----------+--------+------------------+
| Model # | Precision | Recall | Micro : F1-Score |
+---------+-----------+--------+------------------+
| Model 1 |   0.0303  | 0.0129 |      0.0181      |
| Model 2 |   0.3021  | 0.3208 |      0.3111      |
| Model 3 |   0.2722  | 0.4827 |      0.3481      |
| Model 4 |   0.0087  | 0.0051 |      0.0064      |
| Model 5 |   0.2566  | 0.4625 |      0.3301      |
+---------+-----------+--------+------------------+
```

## Conclusion :

1. **The maximum micro averaged F1 score is 0.4015 and the maximum value of recall is 0.4827.**
2. **Char N-gram features proved to be significantly powerful than word N-gram features. Skip Grams were also useful.**
3. **Using featurization like bow, avg word2vec, tfidf word2vec and combination of TF-IDF and Word2Vec features, our models behaved surprisingly better than the previous implementation.**
4. **In today's era, we are more used to see scores above 90%. But given a very limited data size sample of 14K datapoints, we have actually managed to get a decent micro averaged F1 score.**

## References :

**Research Paper:** https://www.aclweb.org/anthology/L18-1274 (https://www.aclweb.org/anthology/L18-1274)

**Code References:** https://www.appliedaicourse.com/ (https://www.appliedaicourse.com/)

**Ideas:** https://en.wikipedia.org/wiki/Multi-label_classification (https://en.wikipedia.org/wiki/Multi-label_classification)

**Binary Relevance:** http://scikit.ml/api/skmultilearn.problem_transform.br.html (http://scikit.ml/api/skmultilearn.problem_transform.br.html)

**Classifiers:** https://scikit-learn.org/stable/ (https://scikit-learn.org/stable/)

**Strategies:** https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/ (https://www.analyticsvidhya.com/blog/2017/08/introduction-to-multi-label-classification/)

**Others :**

https://keras.io/ (https://keras.io/)
https://medium.com/@shivajbd/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e (https://medium.com/@shivajbd/understanding-input-and-output-shape-in-lstm-keras-c501ee95c65e)
https://www.pyimagesearch.com/2019/05/27/keras-feature-extraction-on-large-datasets-with-deep-learning/ (https://www.pyimagesearch.com/2019/05/27/keras-feature-extraction-on-large-datasets-with-deep-learning/)
https://towardsdatascience.com/building-a-multi-label-text-classifier-using-bert-and-tensorflow-f188e0ecdc5d (https://towardsdatascience.com/building-a-multi-label-text-classifier-using-bert-and-tensorflow-f188e0ecdc5d)
https://blog.mimacom.com/text-classification/ (https://blog.mimacom.com/text-classification/)
https://paperswithcode.com/task/multi-label-text-classification (https://paperswithcode.com/task/multi-label-text-classification)
https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/ (https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/)
https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/ (https://machinelearningmastery.com/how-to-calculate-precision-recall-f1-and-more-for-deep-learning-models/)