In [0]:
```
import pandas as pd
import warnings
warnings.filterwarnings("ignore")
import pandas as pd
import sqlite3
import csv
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from wordcloud import WordCloud
import re
import os
from sqlalchemy import create_engine # database connection
import datetime as dt
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.metrics import f1_score,precision_score,recall_score
from sklearn import svm
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from datetime import datetime
```

loading data

In [2]:
```
!pip install -U -q PyDrive
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
```

```python
from oauth2client.client import GoogleCredentials
# Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

```
     |████████████████████████████████| 993kB 6.6MB/s
  Building wheel for PyDrive (setup.py) ... done
```

In [3]:
```python
link = 'https://drive.google.com/open?id=1TUDwODYHxNneCjyss6XhqJaDpB0NrQbG' # The shareable link
fluff, id = link.split('=')
print (id)
```

```
1TUDwODYHxNneCjyss6XhqJaDpB0NrQbG
```

In [0]:
```python
downloaded = drive.CreateFile({'id':id})
downloaded.GetContentFile('mpst_full_data.csv')
```

In [5]:
```python
df=pd.read_csv('mpst_full_data.csv')
df.head()
```

Out[5]:

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

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
4	tt0086250	Scarface	In May 1980, a Cuban man named Tony Montana (A...	cruelty, murder, dramatic, cult, violence, atm...	val	imdb

In [6]:
```
df.shape
```
Out[6]:
```
(14828, 6)
```

preprocessing data

In [0]:
```
sort=df.sort_values('imdb_id')
```

In [8]:
```
f=sort.drop_duplicates(subset={'title','tags','plot_synopsis'})
f.shape
```
Out[8]:
```
(14752, 6)
```

In [9]:
```
final=f.sort_values('split')
final.head()
```
Out[9]:

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
10426	tt6583664	Scapegoat	There are significant differences between this...	revenge, suspenseful	test	wikipedia
10878	tt0076911	What a Nightmare, Charlie Brown!	One winter day, Charlie Brown is trying to pre...	psychedelic	test	wikipedia
1776	tt0499448	The Chronicles of Narnia: Prince Caspian	On a cloudless night in Narnia, under an eclip...	good versus evil, violence, fantasy, boring, r...	test	imdb
335	tt0077235	Big Wednesday	Malibu, California, 1962. Matt the Enforcer (J...	cult, philosophical	test	imdb

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
5293	tt0498399	We Own the Night	New York, November 1988: A new breed of narcot...	revenge, suspenseful, murder, violence, flashback	test	imdb

In [10]:
```
final['split'].value_counts()
```

Out[10]:
```
train    9436
test     2957
val      2359
Name: split, dtype: int64
```

In [11]:
```
final[0:2957].tail()
```

Out[11]:

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
9499	tt0036707	Cheyenne Wildcat	The president of Blue Springs Bank, Jason Hopk...	murder	test	wikipedia
3941	tt1427298	The Human Race	An exciting, unpredictable and emotionally dra...	violence	test	imdb
4092	tt0052901	Horrors of the Black Museum	HORRORS OF THE BLACK MUSEUM In London a delive...	murder	test	imdb
12562	tt1548635	Siren	One week before his wedding day, Jonah and his...	murder	test	wikipedia
10903	tt0050407	Forty Guns	In the 1880s, Griff Bonnell (Barry Sullivan) a...	murder	test	wikipedia

In [12]:
```
final[2957:12393].tail()
```

Out[12]:

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
258	tt0096320	Twins	Julius Benedict and Vincent Benedict are frate...	comedy, entertaining	train	wikipedia

	imdb_id	title	plot_synopsis	tags	split	synopsis_source
10615	tt0095122	The Expendables	The Expendables—leader Barney Ross, knife spec...	cult	train	wikipedia
6280	tt0096018	Running on Empty	Mike (Terry Serio) is a young man who is a bud...	anti war, romantic, avant garde, dramatic	train	wikipedia
312	tt0096734	The 'Burbs	The film starts on a small cul-de-sac suburban...	comedy, mystery, gothic, intrigue, insanity, p...	train	imdb
11311	tt0095626	Messenger of Death	Children play outside a rural Colorado home. T...	good versus evil, revenge, suspenseful, murder...	train	wikipedia

In [0]:
```
stopwords= set(['br', 'the', 'i', 'me', 'my', 'myself', 'we', 'our', 'o
urs', 'ourselves', 'you', "you're", "you've",\
            "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselve
s', 'he', 'him', 'his', 'himself', \
            'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'it
s', 'itself', 'they', 'them', 'their',\
            'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'th
is', 'that', "that'll", 'these', 'those', \
            'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'h
ave', 'has', 'had', 'having', 'do', 'does', \
            'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or',
 'because', 'as', 'until', 'while', 'of', \
            'at', 'by', 'for', 'with', 'about', 'against', 'between',
'into', 'through', 'during', 'before', 'after',\
            'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out',
'on', 'off', 'over', 'under', 'again', 'further',\
            'then', 'once', 'here', 'there', 'when', 'where', 'why', 'h
ow', 'all', 'any', 'both', 'each', 'few', 'more',\
            'most', 'other', 'some', 'such', 'only', 'own', 'same', 's
o', 'than', 'too', 'very', \
            's', 't', 'can', 'will', 'just', 'don', "don't", 'should',
"should've", 'now', 'd', 'll', 'm', 'o', 're', \
            've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't",
```

```
'didn', "didn't", 'doesn', "doesn't", 'hadn',\
            "hadn't", 'hasn', "hasn't", 'haven', "haven't", 'isn', "is
n't", 'ma', 'mightn', "mightn't", 'mustn',\
            "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn',
 "shouldn't", 'wasn', "wasn't", 'weren', "weren't", \
            'won', "won't", 'wouldn', "wouldn't"])
```

In [14]:

```
import nltk
nltk.download('punkt')
from tqdm import tqdm
from bs4 import BeautifulSoup
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer("english")
preprocess_plot=[]
for s in tqdm(final['plot_synopsis'].values):
    s=re.sub(r'S*\d\S*','',s)
    s=re.sub('[^A-Za-z]+',' ',s)
    soup=BeautifulSoup(s)
    s=soup.get_text()
    s=re.sub(r'http\S+','',s)
    words=word_tokenize(s.lower())

    #Removing all single letter and and stopwords from question exceptt
 for the letter 'c'
    s=' '.join(str(stemmer.stem(j)) for j in words if j not in stopword
s and (len(j)!=1 ))
    preprocess_plot.append(s.strip())
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
100%|██████████| 14752/14752 [02:50<00:00, 86.50it/s]
```

In [15]:

```
preprocess_title=[]
for s in tqdm(final['title'].values):
    s=re.sub(r'S*\d\S*','',s)
    s=re.sub('[^A-Za-z]+',' ',s)
    soup=BeautifulSoup(s)
    s=soup.get_text()
```

```
    s=re.sub(r'http\S+','',s)
    words=word_tokenize(s.lower())

    #Removing all single letter and and stopwords from question exceptt
 for the letter 'c'
    s=' '.join(str(stemmer.stem(j)) for j in words if j not in stopword
s and (len(j)!=1 ))
    preprocess_title.append(s.strip())
```

```
100%|██████████| 14752/14752 [00:07<00:00, 2097.70it/s]
```

In [16]:

```
preprocess_tags=[]
for s in tqdm(final['tags'].values):
    s=re.sub(r' ','',s)
    s=re.sub(r',',' ',s)
    s=' '.join(e.lower() for e in s.split() )
    preprocess_tags.append(s.strip())
```

```
100%|██████████| 14752/14752 [00:00<00:00, 199448.06it/s]
```

In [17]:

```
preprocess_plot[1]
```

Out[17]:

```
'one winter day charli brown tri pretend musher snoopi dog idea get cha
rli brown pull fun ride sled night come comfort indoor charli brown ind
ign snoopi adjust well home life remind snoopi fact arctic dog fed day
meal larg consist cold meat raw fish snoopi blanch give look bad come c
onclus snoopi over civil under dogifi dog make scrumptious dinner five
pizza larg milkshak eat snoopi goe bed doghous prompt wake find sled do
g iditarod trail sled dog race alaska presum klondik gold rush serum ru
n nome snoopi cruelli mistreat owner seen shadow silhouett speak much d
eeper version classic peanut adult waa waa waa languag fellow dog run r
ag deni food water dog take turn bark loud snoopi order let know inde o
utsid one scene break snow scene sled master stop honki tonk hungri sno
opi sneak insid snatch sandwich mug root beer sit near piano feign play
washington post march snoopi tri hand game poker keep poker face laugh
loud reveal improb win hand five ace caus brawl leav snoopi escap next
room find stage paint backdrop pari franc cheer danc howev music chang
imperson dancer men throw rotten fruit snoopi thrown bar back sled dog
continu mistreat deni food water unabl take anymor one night snoopi bre
```

```
ak cri done goe convert new life order surviv bare fang fall walk four
snoopi challeng lead dog fight win becom alpha male sled dog pack also
turn tabl rest dog deni food water eventu lead owner ice cover lake ice
crack caus sled dog owner swallow water snoopi find pull hole sink scre
am life snoopi wake cling side doghous reliev nightmar snoopi wake char
li brown recount nightmar pantomim charli brown allow snoopi spend nigh
t insid not snoopi help larg ice cream sunda remind arctic experi inde
nightmar'
```

In [18]:
```
preprocess_tags[2]
```

Out[18]:
```
'goodversusevil violence fantasy boring romantic'
```

In [19]:
```
preprocess_tags[5]
```

Out[19]:
```
'boring murder dramatic violence flashback claustrophobic sentimental'
```

choosing correct tags and vectorizing them

In [0]:
```
vectorizer = CountVectorizer(tokenizer = lambda x: x.split(), binary='t
rue')
multilabel_y = vectorizer.fit_transform(preprocess_tags)
```

In [21]:
```
multilabel_y.shape
```

Out[21]:
```
(14752, 71)
```

train cv and test split

In [0]:
```
x_train1=preprocess_plot[2957:12393]
x_test1=preprocess_plot[:2957]
x_cv1=preprocess_plot[12393:]

x_train2=preprocess_title[2957:12393]
```

```
x_test2=preprocess_title[:2957]
x_cv2=preprocess_title[12393:]

y_train = multilabel_y[2957:12393]
y_test = multilabel_y[:2957]
y_cv=   multilabel_y[12393:]
```

200 dim word2vec of plot

In [0]:
```
from gensim.models import Word2Vec
from sklearn.feature_extraction.text import TfidfVectorizer
trlst=[]
for s in x_train1:
    trlst.append(s.split())
```

In [0]:
```
w2vtr=Word2Vec(trlst,size=200,min_count=5, workers=4)
```

In [0]:
```
w2v_wordstr = list(w2vtr.wv.vocab)
```

In [0]:
```
model= TfidfVectorizer(min_df=10)
tfidf=model.fit_transform(x_train1)
dictonary=dict(zip(model.get_feature_names(),model.idf_))
```

In [0]:
```
tr=[]
x= model.get_feature_names()
for s in tqdm(x_train1):
    a=s.split()
    vec= np.zeros(200)
    sum=0
    for b in a:
        if b in w2v_wordstr and b in x:
            tfidf=dictonary[b]*(a.count(b)/len(b))
            vec+=w2vtr.wv[b]*tfidf
            sum+=tfidf
    if sum != 0:
```

```
        vec=vec/sum

    tr.append(vec)
```

```
100%|██████████| 9436/9436 [23:37<00:00,  6.66it/s]
```

In [0]:

```
cv=[]
for s in tqdm(x_cv1):
    a=s.split()
    vec= np.zeros(200)
    sum=0
    for b in a:
        if b in w2v_wordstr and b in x:
            tfidf=dictonary[b]*(a.count(b)/len(b))
            vec+=w2vtr.wv[b]*tfidf
            sum+=tfidf
    if sum != 0:
        vec=vec/sum

    cv.append(vec)
```

```
100%|██████████| 2359/2359 [06:11<00:00,  7.60it/s]
```

In [0]:

```
test=[]
for s in tqdm(x_test1):
    a=s.split()
    vec= np.zeros(200)
    sum=0
    for b in a:
        if b in w2v_wordstr and b in x:
            tfidf=dictonary[b]*(a.count(b)/len(b))
            vec+=w2vtr.wv[b]*tfidf
            sum+=tfidf
    if sum != 0:
        vec=vec/sum
```

```
    test.append(vec)
```

```
100%|██████████| 2957/2957 [07:31<00:00,  6.51it/s]
```

tfidf

In [0]:
```
vectorizer =TfidfVectorizer(min_df=10,ngram_range=(1,3),max_features=20
000)
x_train_multilabel1 = vectorizer.fit_transform(x_train1)
x_cv_multilabel1 = vectorizer.transform(x_cv1)
x_test_multilabel1 = vectorizer.transform(x_test1)
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabel1.shape, "Y :",y_
train.shape)
print("Dimensions of cv data X:",x_cv_multilabel1.shape, "Y :",y_cv.sha
pe)
print("Dimensions of test data X:",x_test_multilabel1.shape,"Y:",y_test
.shape)
```

```
Dimensions of train data X: (9436, 20000) Y : (9436, 71)
Dimensions of cv data X: (2359, 20000) Y : (2359, 71)
Dimensions of test data X: (2957, 20000) Y: (2957, 71)
```

fourgram of plot

In [0]:
```
vectorizer =TfidfVectorizer(min_df=5,ngram_range=(4,4),max_features=500
0)
x_train_multilabel2 = vectorizer.fit_transform(x_train1)
x_cv_multilabel2 = vectorizer.transform(x_cv1)
x_test_multilabel2 = vectorizer.transform(x_test1)
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabel2.shape, "Y :",y_
train.shape)
```

```
print("Dimensions of cv data X:",x_cv_multilabel2.shape, "Y :",y_cv.sha
pe)
print("Dimensions of test data X:",x_test_multilabel2.shape,"Y:",y_test
.shape)
```

```
Dimensions of train data X: (9436, 3590) Y : (9436, 71)
Dimensions of cv data X: (2359, 3590) Y : (2359, 71)
Dimensions of test data X: (2957, 3590) Y: (2957, 71)
```

unigram and bigram of title

In [0]:
```
vectorizer =TfidfVectorizer(ngram_range=(1,2),max_features=10000)
x_train_multilabel3 = vectorizer.fit_transform(x_train2)
x_cv_multilabel3 = vectorizer.transform(x_cv2)
x_test_multilabel3 = vectorizer.transform(x_test2)
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabel3.shape, "Y :",y_
train.shape)
print("Dimensions of cv data X:",x_cv_multilabel3.shape, "Y :",y_cv.sha
pe)
print("Dimensions of test data X:",x_test_multilabel3.shape,"Y:",y_test
.shape)
```

```
Dimensions of train data X: (9436, 10000) Y : (9436, 71)
Dimensions of cv data X: (2359, 10000) Y : (2359, 71)
Dimensions of test data X: (2957, 10000) Y: (2957, 71)
```

char gram of plot

In [0]:
```
vectorizer =TfidfVectorizer(min_df=10,analyzer='char',ngram_range=(4,6
),max_features=20000)
x_train_multilabel4 = vectorizer.fit_transform(x_train1)
x_cv_multilabel4 = vectorizer.transform(x_cv1)
x_test_multilabel4 = vectorizer.transform(x_test1)
```

matrix factorization using nmf

In [0]:
```
vectorizer =CountVectorizer(min_df=10)
x_train_multilabl6 = vectorizer.fit_transform(x_train1)
x_cv_multilabl6 = vectorizer.transform(x_cv1)
x_test_multilabl6 = vectorizer.transform(x_test1)
```

In [0]:
```
print("Dimensions of train data X:",x_train_multilabl6.shape, "Y :",y_t
rain.shape)
print("Dimensions of cv data X:",x_cv_multilabl6.shape, "Y :",y_cv.shap
e)
print("Dimensions of test data X:",x_test_multilabl6.shape,"Y:",y_test.
shape)
```

```
Dimensions of train data X: (9436, 13780) Y : (9436, 71)
Dimensions of cv data X: (2359, 13780) Y : (2359, 71)
Dimensions of test data X: (2957, 13780) Y: (2957, 71)
```

In [0]:
```
from sklearn.decomposition import NMF
nmf=NMF(n_components=200)
tr1=nmf.fit_transform(x_train_multilabl6)
cv1=nmf.transform(x_cv_multilabl6)
test1=nmf.transform(x_test_multilabl6)
```

(unigram+bigram+trigram+fourgram+word2vec) of plot+unigram and bigram of title+nmf of plot

In [0]:
```
from scipy.sparse import hstack
x_train_multilabl = hstack([x_train_multilabel1,x_train_multilabel2,x_t
rain_multilabel3,x_train_multilabel4,tr,tr1])
print(x_train_multilabl.shape)
x_cv_multilabl = hstack([x_cv_multilabel1,x_cv_multilabel2,x_cv_multila
bel3,x_cv_multilabel4,cv,cv1])
print(x_cv_multilabl.shape)
```

```
x_test_multilabl = hstack([x_test_multilabel1,x_test_multilabel2,x_test_multilabel3,x_test_multilabel4,test,test1])
print(x_test_multilabl.shape)
```

```
(9436, 54000)
(2359, 54000)
(2957, 54000)
```

logistic regression

In [0]:

```
start = datetime.now()
alpha=[0.01,0.1,1,10]
for j in alpha:
    classifier_1 = OneVsRestClassifier(LogisticRegression(C=j,penalty='l1',tol=0.001))
    classifier_1.fit(x_train_multilabl, y_train)
    y_pred_prob = classifier_1.predict_proba(x_train_multilabl)
    i=0.2
    predictions_1 = (y_pred_prob >= i).astype(int)
    y_pred_probc = classifier_1.predict_proba(x_cv_multilabl)

    predictions_2 = (y_pred_probc >= i).astype(int)

    f1 = f1_score(y_train, predictions_1, average='micro')
    f2 = f1_score(y_cv, predictions_2, average='micro')
    print("Micro-average quality numbers for C=",j)
    print(" F1-measure for train: {:.4f}".format( f1))
    print(" F1-measure for cv: {:.4f}".format( f2))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Micro-average quality numbers for C= 0.01
 F1-measure for train: 0.3378
 F1-measure for cv: 0.3405
Micro-average quality numbers for C= 0.1
 F1-measure for train: 0.3867
 F1-measure for cv: 0.3786
Micro-average quality numbers for C= 1
```

```
 F1-measure for train: 0.5037
 F1-measure for cv: 0.4015
Micro-average quality numbers for C= 10
 F1-measure for train: 0.9620
 F1-measure for cv: 0.3400
Time taken to run this cell : 0:38:36.247697
```

decreasing threshold on test

In [0]:
```
classifier_1 = OneVsRestClassifier(LogisticRegression(C=1,penalty='l2',
tol=0.001))
classifier_1.fit(x_train_multilabl, y_train)
y_pred_prob = classifier_1.predict_proba(x_test_multilabl)
i=0.2
y_pred_new = (y_pred_prob >= i).astype(int)
f2 = f1_score(y_test, y_pred_new, average='micro')
print("Micro-average quality numbers for C=0.01 and threshold=",i)
print(" F1-measure for test: {:.4f}".format( f2))
```

```
Micro-average quality numbers for C=0.01 and threshold= 0.2
 F1-measure for test: 0.3979
```

naive bayes

In [0]:
```
x_train_multilabl = hstack([x_train_multilabel1,x_train_multilabel2,x_t
rain_multilabel3,tr,tr1,tr2])
print(x_train_multilabl.shape)
x_cv_multilabl = hstack([x_cv_multilabel1,x_cv_multilabel2,x_cv_multila
bel3,cv,cv1,cv2])
print(x_cv_multilabl.shape)

x_test_multilabl = hstack([x_test_multilabel1,x_test_multilabel2,x_test
_multilabel3,test,test1,test2])
print(x_test_multilabl.shape)
```

```
(9436, 34000)
```

```
(2359, 34000)
(2957, 34000)
```

In [0]:

```
from sklearn.naive_bayes import BernoulliNB
start = datetime.now()
alpha=[0.001,0.01,0.1,1,10]
for i in alpha:
    classifier_1 = OneVsRestClassifier(BernoulliNB(alpha=i))
    classifier_1.fit(x_train_multilabl, y_train)
    predictions_1 = classifier_1.predict(x_train_multilabl)
    predictions_2 = classifier_1.predict(x_cv_multilabl)
    f1 = f1_score(y_train, predictions_1, average='micro')
    f2 = f1_score(y_cv, predictions_2, average='micro')
    print("Micro-average quality numbers for C=",i)
    print(" F1-measure for train: {:.4f}".format( f1))
    print(" F1-measure for cv: {:.4f}".format( f2))
print("Time taken to run this cell :", datetime.now() - start)
```

```
Micro-average quality numbers for C= 0.001
 F1-measure for train: 0.6216
 F1-measure for cv: 0.3447
Micro-average quality numbers for C= 0.01
 F1-measure for train: 0.5674
 F1-measure for cv: 0.3393
Micro-average quality numbers for C= 0.1
 F1-measure for train: 0.4540
 F1-measure for cv: 0.3024
Micro-average quality numbers for C= 1
 F1-measure for train: 0.3152
 F1-measure for cv: 0.2559
Micro-average quality numbers for C= 10
 F1-measure for train: 0.2087
 F1-measure for cv: 0.1847
Time taken to run this cell : 0:04:25.800332
```

In [0]:

```
classifier_1 = OneVsRestClassifier(BernoulliNB(alpha=0.001))
classifier_1.fit(x_train_multilabl, y_train)
y_pred_prob = classifier_1.predict_proba(x_test_multilabl)
```

```
y_pred_new = (y_pred_prob >= 0.01).astype(int)
f2 = f1_score(y_test, y_pred_new, average='micro')
print("Micro-average quality numbers for C=0.001 ")
print(" F1-measure for test: {:.4f}".format( f2))
```

```
Micro-average quality numbers for C=0.001
 F1-measure for test: 0.3454
```

using deep learning technique

In [0]:
```
count_vect = CountVectorizer(min_df=10) #in scikit-learn CountVectorize
r(min_df=10, max_features=500)
X=count_vect.fit_transform(x_train1)
top_feat=count_vect.get_feature_names()
```

In [0]:
```
X.shape
```

Out[0]:
```
(9436, 13780)
```

In [0]:
```
#placing  words in reviews in a list
word=[]
for sent in x_train1:
    wrds = sent.split()
    for wrd in wrds:
        word.append(wrd)
```

In [0]:
```
a=np.arange(13780)
feat10k=dict( zip( top_feat, a))
```

In [0]:
```
feat=np.zeros(13780,dtype='int')
for wrd in word:
  if feat10k.get(wrd,-1)>0:
    feat[feat10k[wrd]]+=1
```

In [0]:
```
sort_feat=np.argsort(feat)
sorted_list=[]
for i in sort_feat:
    sorted_list.append(top_feat[i])
```

In [0]:
```
sort_freq=dict( zip( sorted_list, sort_feat))
```

In [0]:
```
ranktr=[]
for sent in x_train1:
    lst=[]
    wrds = sent.split()
    for wrd in wrds:
        if feat10k.get(wrd,-1)>0:
            lst.append(sort_freq[wrd])
    ranktr.append(lst)
```

In [0]:
```
rankcv=[]
for sent in x_cv1:
    lst=[]
    wrds = sent.split()
    for wrd in wrds:
        if feat10k.get(wrd,-1)>0:
            lst.append(sort_freq[wrd])
    rankcv.append(lst)
```

In [0]:
```
ranktest=[]
for sent in x_test1:
    lst=[]
    wrds = sent.split()
    for wrd in wrds:
        if feat10k.get(wrd,-1)>0:
            lst.append(sort_freq[wrd])
    ranktest.append(lst)
```

In [0]:
```
a=max(x_train1, key=len).strip()
```

In [0]:

```
from keras.models import Sequential
from keras.layers import Dense
from keras.layers import LSTM
from keras.layers import Bidirectional
from keras.layers.embeddings import Embedding
from keras.preprocessing import sequence
# fix random seed for reproducibility
np.random.seed(7)
```

```
Using TensorFlow backend.
```

In [0]:

```
max_review_length = 1000
X_train = sequence.pad_sequences(ranktr, maxlen=max_review_length)
X_cv   = sequence.pad_sequences(rankcv, maxlen=max_review_length)
X_test = sequence.pad_sequences(ranktest, maxlen=max_review_length)

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

```
(9436, 1000)
[     0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
      0      0      0      0      0      0      0      0      0      0      0
0
```

```
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0     0
0
     0     0     0     0     0     0     0     0     0     0  4518  860
3
  4693  2203  7121 13621  3694 12193  4396  6686  6038 10776  2120 1222
7
  1555 13349  6414  2007 13450 11010  1868  2203  8320 12921 11674  439
6
 10119  1555  3047  7996  7235  8297  9832   220  6967   885  1433  220
3
   721  6686  9832 11616  6983 10776  7082  3339  5079  7121  9677  143
1
  7051 13349   826 13539 13462 13349 13450 11010  1813  6058   623  486
0
  3227 10044  1365  1735 10445 13299  8151  5709  4493    69  4194  455
6
  4518  9361 13349 10951  6733  2756 11111 13349  9906  8873  2593  967
7
  1431  7121  7051 12227  4396  3047 12542  8159  7211 13349 12600  832
4
 13734  2203  4529  8562  8901 10445  8768 13557  8277 10962  5141  699
```

```
3
  4556  4518  1683 12397 10445  6993  1506 11751  6945  4038  1538 1038
2
  4349  6983 11528  5141  6331 13764  5859 13317  8892 12418  5058   72
3
  9582  8964 12654  4041  7646  8324 13734   625 10784  3024 10445  692
0
  2265 13349  4510  8477  8324 13734  9305  8541 10445 11025  7211  278
4
 13401 13349 12227 10445  4396 10445 12043  8495  5646  9918 10776  771
3
  1526  8884  9832  5079 13632 12672  5122  2120  4193  9456  5053  970
4
  3997 12406  8884  4529    94   526  6258  7211 13349  3768 13387 1176
7
 10164  8477 11816 12383  6094 13738  1434  8213  2040     2  6546  429
0
   982  9918  6399   624  9305 10735 13711  6239   637  3654 12347  456
3
 12300 12749  3115  4609 11816  4926  5079 10445 13387  2040 11767 1222
7
 13349  7121   334 10735 13711   723  6740  7235  9832  3721  7226  712
1
  3654  9860 10445  8460  4077  7713    99 10445 10445 12629 13047  503
6
  7121 10046  8442 13635  7691  9832  4529  8442   321  1863 13693  215
1
  3596 13721 11332 13401 12749  4228  9711  2629  9832 11277  2956  674
0
  6686  6038 10776 11417  6919 10445 13349 13347  7211 10811  1585  881
4
  4022  2040  7048  4006  5865  8397  2040  1520   726  8397  6336  490
8
  1898 10445 13349  3443 10445  7582  7420  4992 11011  5786  8213  323
6
  4325  2288 13349  8320 12728  8539 12629 11570  3330 13349 10445  467
4
  7235  9832  9507  7076  4074  8442  4529 11277 10801  8908 13349  342
5
```

```
  4551 10445  8524  2652 13349  4074 10801 10398  2593 12610 12376  152
6
  4001  9832  5321   721 10445 10145  7121  4142 10735 13711 12364 1081
1
  9832  5443  6038 10776 13693  2151   966  7160  1555  3997  2120  983
2
 10811 10896  4541  1603 13738  2040 11067  7421 10776  7121  4142 1039
3
 13349 10445  2418  9832  9493   770  8588 12356    99  4249  8588  746
6
   398  8114 13539 10270  2889  2782 12822  1814  2782 10445 10918  436
7
  6340  7121  3083 11668  9669  8127 13349  3465  7121  4529  5079 1027
0
 11689 10485  6206 13349 12227  7121 10445 13179  9518 10445  8732  712
1
  8442  4622 10445  3284  1154 13349  5880  9711  7421  2634 10907  445
0
  9582 10916  9175  4510 10445  7121 12001 13325  4489  5596 11125 1044
5
 12126  8892 11005  4529 12007  9832 11593  1433   543 12364 12127  275
1
  9159 12402  6993  3122 13349 10445  4524  4022 13693  2151 10798  241
8
  7161 12467 13693  1433 13693  2151 12402 12629  7188 10776  8442  679
7
  2040 12127 12227  5213 10697  9832 13659  6740  1008 10776 10197  639
9
 13693  2151 13659 13047 10776 12921  4034  2120  2211 12373 10197 1369
3
  2151 13659 13047  4967  5349  3479 10446  4357  6983  9832 12256  712
9
 12373  6755 13349 10445  4142 12542   637  3654  3285  6332 12206  173
5
  3104  9881 13047  4529  2040  5702  7363 13400  8170  9918 10776 1212
7
  1433 12364   624 12626  2040 10202  4862  8213  2040  2022   472   39
8
  8114    99 10896 13715  9501  2022  1215  1474  5122  9704 13187   87
```

```
0
 10446  4357 11987  4057 13349  4529 13693  2151  7121 13450  4554    92
2
  6740  7161 12992  7121  3464 13693  9219  5880 13693  9595    65   756
4
   643 11999 13349 11207  7121   625 12418  6740  4554 10355 10652  1334
9
   637 10445  4754 13401  5865  8884  2529  9832  9476  4142   637    62
4
  9832 11044 11062 10445 13349  9669  9832  1183  1091  4039 12076   451
0
  9832  9672 10445  5090  4845  2288 10445  1912  8588  2288  5443   983
2
  8263  3115 13349 13349  9760  3047  4396  4350  2483  6967 11921   522
8
  9832 10849  4616  2288  4325  7298 10445  1912 13349  9832  9272   305
8
 10445 13349  3083  6613  5289  5772  6904  1526  4577  1183  1091   687
7
  1863  2593  9704 13187 10445 13349  6789   637 13522  4862  8213  1117
8
   636  2485  3654   637  3083  1027 13685 11782 10221   618  3104   988
1
 12206   663 10445 13047 11011  5786  8213 10445  7121  6770 10445   957
1
 13349  5122  5785  5036  5303  4635  8089  9155  8089  9155 12356   632
7
  5785  1020  1971  8114  7421 12345 13715  8114   472   398 10376   160
0
  2040 11791   797 13349   334  7453  8603  1431  4396 10861  4529   777
9
  6229 10075  6049  4357]
```

In [0]:

```
embedding_vecor_length = 300
model_1 = Sequential()
model_1.add(Embedding(13780+1, embedding_vecor_length, input_length=max
_review_length))
model_1.add(LSTM(128))
model_1.add(Dense(71, activation='sigmoid'))
```

```
model_1.compile(loss='binary_crossentropy', optimizer='adam', metrics=[
'accuracy'])
```

```
WARNING: Logging before flag parsing goes to stderr.
W0817 10:11:53.313753 140710187665280 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen
d.py:74: The name tf.get_default_graph is deprecated. Please use tf.com
pat.v1.get_default_graph instead.

W0817 10:11:53.354623 140710187665280 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen
d.py:517: The name tf.placeholder is deprecated. Please use tf.compat.v
1.placeholder instead.

W0817 10:11:53.361517 140710187665280 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen
d.py:4138: The name tf.random_uniform is deprecated. Please use tf.rand
om.uniform instead.

W0817 10:11:53.683286 140710187665280 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/optimizers.py:790: The nam
e tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optim
izer instead.

W0817 10:11:53.710934 140710187665280 deprecation_wrapper.py:119] From
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen
d.py:3376: The name tf.log is deprecated. Please use tf.math.log instea
d.

W0817 10:11:53.720319 140710187665280 deprecation.py:323] From /usr/loc
al/lib/python3.6/dist-packages/tensorflow/python/ops/nn_impl.py:180: ad
d_dispatch_support.<locals>.wrapper (from tensorflow.python.ops.array_o
ps) is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
```

In [0]:
```
model_1.fit(X_train, y_train, epochs=5, batch_size=512,validation_data=
(X_cv,y_cv))
```

```
W0817 10:11:54.956021 140710187665280 deprecation_wrapper.py:119] From
```

```
/usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backen
d.py:986: The name tf.assign_add is deprecated. Please use tf.compat.v
1.assign_add instead.
```

```
Train on 9436 samples, validate on 2359 samples
Epoch 1/5
9436/9436 [==============================] - 51s 5ms/step - loss: 0.548
5 - acc: 0.8645 - val_loss: 0.2306 - val_acc: 0.9574
Epoch 2/5
9436/9436 [==============================] - 47s 5ms/step - loss: 0.164
5 - acc: 0.9583 - val_loss: 0.1449 - val_acc: 0.9574
Epoch 3/5
9436/9436 [==============================] - 46s 5ms/step - loss: 0.141
1 - acc: 0.9583 - val_loss: 0.1421 - val_acc: 0.9574
Epoch 4/5
9436/9436 [==============================] - 46s 5ms/step - loss: 0.139
3 - acc: 0.9583 - val_loss: 0.1412 - val_acc: 0.9574
Epoch 5/5
9436/9436 [==============================] - 46s 5ms/step - loss: 0.138
7 - acc: 0.9583 - val_loss: 0.1410 - val_acc: 0.9574
```

Out[0]:

```
<keras.callbacks.History at 0x7ff96ea0ab70>
```

In [0]:

```
predict_probtr = model_1.predict(X_train)
```

In [0]:

```
t=[0.5,0.2,0.1,0.05]#threshold
```

In [0]:

```
for i in t:
  y_predtr = (predict_probtr >= i).astype(int)
  f1 = f1_score(y_train, y_predtr, average='micro')
  print(" F1-measure for train: {:.4f} and threshold={}".format( f1,i))
```

```
 F1-measure for train: 0.0000 and threshold=0.5
 F1-measure for train: 0.2773 and threshold=0.2
 F1-measure for train: 0.3053 and threshold=0.1
 F1-measure for train: 0.2485 and threshold=0.05
```

taking threshold as 0.1

In [0]:
```
predict_prob = model_1.predict(X_test)
```

In [0]:
```
predict_prob[0]
```

Out[0]:
```
array([0.01858133, 0.04362813, 0.00821382, 0.00974792, 0.00627637,
       0.01454931, 0.00741696, 0.02862522, 0.00422007, 0.01805571,
       0.00680026, 0.01417279, 0.0356462 , 0.00719705, 0.0037584 ,
       0.00538141, 0.00522521, 0.12843624, 0.00783843, 0.02855575,
       0.17216185, 0.01242733, 0.02835459, 0.01230884, 0.02762339,
       0.04805008, 0.03844845, 0.00480804, 0.19711196, 0.06184345,
       0.03073347, 0.00541219, 0.00966036, 0.01791635, 0.01042733,
       0.00973195, 0.03316283, 0.05565515, 0.04421234, 0.00669426,
       0.01055536, 0.00592789, 0.02996501, 0.37207168, 0.03519765,
       0.05066758, 0.00272527, 0.03214604, 0.01680574, 0.0126853 ,
       0.0106259 , 0.01619208, 0.13411337, 0.01968622, 0.00567257,
       0.01547539, 0.16526577, 0.18440023, 0.04246202, 0.05527481,
       0.02168334, 0.01704258, 0.02548701, 0.01247486, 0.00274321,
       0.07060099, 0.00799596, 0.03832561, 0.28975213, 0.00473303,
       0.00628313], dtype=float32)
```

In [0]:
```
y_pred_new = (predict_prob >= 0.1).astype(int)
f2 = f1_score(y_test, y_pred_new, average='micro')
print(" F1-measure for test: {:.4f}".format( f2))
```

```
 F1-measure for test: 0.3100
```

In [0]:
```
print(y_test)
```

```
  (0, 56)	1
  (0, 65)	1
  (1, 52)	1
  (2, 29)	1
  (2, 68)	1
  (2, 26)	1
  (2, 12)	1
```

```
(2, 57)       1
(3, 20)       1
(3, 48)       1
(4, 56)       1
(4, 65)       1
(4, 68)       1
(4, 43)       1
(4, 28)       1
(5, 68)       1
(5, 12)       1
(5, 43)       1
(5, 28)       1
(5, 24)       1
(5, 15)       1
(5, 61)       1
(6, 52)       1
(6, 68)       1
(6, 43)       1
:      :
(2940, 43)    1
(2941, 65)    1
(2942, 43)    1
(2943, 47)    1
(2944, 56)    1
(2944, 65)    1
(2944, 68)    1
(2944, 45)    1
(2945, 43)    1
(2945, 38)    1
(2946, 7)     1
(2947, 28)    1
(2947, 17)    1
(2947, 47)    1
(2947, 37)    1
(2948, 12)    1
(2949, 56)    1
(2949, 57)    1
(2950, 47)    1
(2951, 43)    1
```

```
  (2952, 43)	1
  (2953, 68)	1
  (2954, 43)	1
  (2955, 43)	1
  (2956, 43)	1
```

checking the predicted values

In [0]:
```
y_pred_new[2][57]
```

Out[0]:
```
1
```

In [0]:
```
y_pred_new[2][68]
```

Out[0]:
```
1
```

In [0]:
```
y_pred_new[2]
```

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

out of 4 tags in actual dataset at y_test[2] this model is predicting 2 correctly

the best metrics will be to check recall value.i.e how many times we are actually predicting correct label

In [0]:
```
f2 = recall_score(y_test, y_pred_new, average='micro')
```

In [0]:
```
print(" recall-measure for test: {:.4f}".format( f2))
```

```
 recall-measure for test: 0.5620
```

let check precision also

In [0]:
```
f2 = precision_score(y_test, y_pred_new, average='micro')
```

In [0]:
```
print(" precision-measure for test: {:.4f}".format( f2))
```

```
 precision-measure for test: 0.2140
```

the recall value is great.approx half of the actual value we are able to recall

TOPIC MODELLING

for using randomised search cv and 3 fold cv we merge train and cv together

In [0]:
```
x_train1=preprocess_plot[2957:]
x_test1=preprocess_plot[:2957]

y_train = multilabel_y[2957:]
y_test = multilabel_y[:2957]
```

In [0]:
```
vectorizer =TfidfVectorizer(min_df=10,ngram_range=(1,3),max_features=20
000)
x_train_multilabel1 = vectorizer.fit_transform(x_train1)
x_test_multilabel1 = vectorizer.transform(x_test1)
```

In [28]:
```
x_train_multilabel1.shape
```
Out[28]:
```
(11795, 20000)
```

reference: https://www.analyticsvidhya.com/blog/2016/08/beginners-guide-to-topic-modeling-in-python/

In [0]:
```
from sklearn.decomposition import LatentDirichletAllocation
lda=LatentDirichletAllocation()
tr2=lda.fit_transform(x_train_multilabel1)
test2=lda.transform(x_test_multilabel1)
```

In [29]:
```
from scipy.sparse import hstack
x_train_multilabl = hstack([x_train_multilabel1,tr2])
print(x_train_multilabl.shape)
x_test_multilabl = hstack([x_test_multilabel1,test2])
print(x_test_multilabl.shape)
```

```
(11795, 20010)
(2957, 20010)
```

logistic regression

In [0]:
```
# I am keeping same hyperparameter which i got after tuning above
classifier_1 = OneVsRestClassifier(LogisticRegression(C=1,penalty='l2',
tol=0.001))
classifier_1.fit(x_train_multilabl, y_train)
yprobtr= classifier_1.predict_proba(x_train_multilabl)
y_pred_prob = classifier_1.predict_proba(x_test_multilabl )
i=0.2
y_predtr = (yprobtr >= i).astype(int)
y_pred_new = (y_pred_prob >= i).astype(int)
f1 = f1_score(y_train,y_predtr, average='micro')
f2 = f1_score(y_test, y_pred_new, average='micro')
print("Micro-average quality numbers for C=1 and threshold=",i)
print(" F1-measure for test: {:.4f}".format( f1))
print(" F1-measure for test: {:.4f}".format( f2))
```

```
Micro-average quality numbers for C=1 and threshold= 0.2
 F1-measure for test: 0.5466
 F1-measure for test: 0.4020
```

LightGBM

In [0]:
```
import lightgbm as lgb
from sklearn.model_selection import RandomizedSearchCV
dtr =  OneVsRestClassifier(lgb.LGBMClassifier())
prams={

     'estimator__num_iterations':[100,200,500,1000,2000],
     'estimator__max_depth':[3,5,10,30]
}
random_cfl=RandomizedSearchCV(dtr,param_distributions=prams,scoring='f1
_micro',verbose=10,n_jobs=-1)
random_cfl.fit(x_train_multilabel1,y_train)
```

```
Fitting 3 folds for each of 10 candidates, totalling 30 fits
```

```
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 2 concurrent work
ers.
[Parallel(n_jobs=-1)]: Done   1 tasks      | elapsed:  8.1min
[Parallel(n_jobs=-1)]: Done   4 tasks      | elapsed: 29.6min
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed: 92.5min
[Parallel(n_jobs=-1)]: Done  14 tasks      | elapsed: 117.0min
[Parallel(n_jobs=-1)]: Done  21 tasks      | elapsed: 168.6min
[Parallel(n_jobs=-1)]: Done  30 out of  30 | elapsed: 215.3min finished
```

Out[0]:
```
RandomizedSearchCV(cv='warn', error_score='raise-deprecating',
                   estimator=OneVsRestClassifier(estimator=LGBMClassifi
er(boosting_type='gbdt',

   class_weight=None,

   colsample_bytree=1.0,
```

```
importance_type='split',

learning_rate=0.1,

max_depth=-1,

min_child_samples=20,

min_child_weight=0.001,

min_split_gain=0.0,

n_estimators=100,

n_jobs=-1,

num_leaves=31,

objective=None,...

reg_alpha=0.0,

reg_lambda=0.0,

silent=True,

subsample=1.0,

subsample_for_bin=200000,

subsample_freq=0),
                                              n_jobs=None),
                iid='warn', n_iter=10, n_jobs=-1,
                param_distributions={'estimator__max_depth': [3, 5,
10, 30],
                                     'estimator__num_iterations': [1
00, 200,
                                                                   5
00, 1000,
```

```
000]},
                   pre_dispatch='2*n_jobs', random_state=None, refit=Tr
ue,
                   return_train_score=False, scoring='f1_micro', verbos
e=10)
```

In [0]:
```
random_cfl.best_params_
```

Out[0]:
```
{'estimator__max_depth': 3, 'estimator__num_iterations': 500}
```

In [30]:
```
import lightgbm as lgb
dtr =  OneVsRestClassifier(lgb.LGBMClassifier(max_depth=3,num_iteration
s=500))
dtr.fit(x_train_multilabl, y_train)
```

Out[30]:
```
OneVsRestClassifier(estimator=LGBMClassifier(boosting_type='gbdt',
                                             class_weight=None,
                                             colsample_bytree=1.0,
                                             importance_type='split',
                                             learning_rate=0.1, max_dep
th=3,
                                             min_child_samples=20,
                                             min_child_weight=0.001,
                                             min_split_gain=0.0,
                                             n_estimators=100, n_jobs=-
1,
                                             num_iterations=500, num_le
aves=31,
                                             objective=None, random_sta
te=None,
                                             reg_alpha=0.0, reg_lambda=
0.0,
                                             silent=True, subsample=1.
0,
                                             subsample_for_bin=200000,
                                             subsample_freq=0),
                    n_jobs=None)
```

In [0]:
```
ytrprob=dtr.predict_proba(x_train_multilabl)
```

In [0]:
```
ytestprob=dtr.predict_proba(x_test_multilabl)
```

In [33]:
```
threshold=[0.5,0.3,0.2,0.1]
for i in threshold:
  y_predtr = (ytrprob >= i).astype(int)
  y_predtest = (ytestprob >= i).astype(int)
  f1 = f1_score(y_train,y_predtr, average='micro')
  f2 = f1_score(y_test, y_predtest, average='micro')
  print("Micro-average quality numbers for C=1 and threshold=",i)
  print(" F1-measure for test: {:.4f}".format( f1))
  print(" F1-measure for test: {:.4f}".format( f2))
```

```
Micro-average quality numbers for C=1 and threshold= 0.5
 F1-measure for test: 0.8106
 F1-measure for test: 0.2981
Micro-average quality numbers for C=1 and threshold= 0.3
 F1-measure for test: 0.8812
 F1-measure for test: 0.3861
Micro-average quality numbers for C=1 and threshold= 0.2
 F1-measure for test: 0.8380
 F1-measure for test: 0.4087
Micro-average quality numbers for C=1 and threshold= 0.1
 F1-measure for test: 0.6569
 F1-measure for test: 0.3902
```

In [1]:
```
from prettytable import PrettyTable

# Initializing table object
x = PrettyTable()

x.field_names = ["vectorizer","Model","train f1","test f1"]
x.add_row(["tfidf","logistic regression","0.4015","0.3979" ])
x.add_row(["tfidf","naive bayes","0.3447","0.3454" ])
x.add_row(["deep learning","LSTM","0.3173","0.3215" ])
x.add_row(["topic medelling+ tfidf","Logistic Regression","0.54","0.40"
 ])
```

```
x.add_row(["topic medelling+ tfidf","LightGBM","0.83","0.408" ])
print(x)
```

```
+-----------------------+--------------------+----------+---------+
|       vectorizer      |       Model        | train f1 | test f1 |
+-----------------------+--------------------+----------+---------+
|         tfidf         | logistic regression |  0.4015  |  0.3979 |
|         tfidf         |     naive bayes    |  0.3447  |  0.3454 |
|     deep learning     |        LSTM        |  0.3173  |  0.3215 |
| topic medelling+ tfidf | Logistic Regression |   0.54   |   0.40  |
| topic medelling+ tfidf |      LightGBM      |   0.83   |  0.408  |
+-----------------------+--------------------+----------+---------+
```

conclusion

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

1. I got a significant improvement of around 5% from original research paper mentioned in above link.
2. Experimenting with lots of featurization and including topic modelling improved my score.
3. Hacks using threshold was very effecient and improved score.
4. tags need to be preprocessed as it contained multiple duplicate tags with a space in front.after preprocessing i got 71 tags outof 141
5. I used every possible approach i.e tfidf,matrix factorization techniques and deep learning.
6. The final best micro f1 on test was .408.
7. The given dataset was very small and plot featurization increased dimention . due to lack of sufficient data point deep learning didnt perform well.
8. I trained model with train and test split specified by the dataset.
9. I cared about data leakage.
10. Decision trees was the worst performer because of high dimention.