PROJECT AS PART OF CS6109 - COMPILER DESIGN

TOPIC : Movie Genre Classification Based on Semantic Analysis

Presented by :

Aaaksh K - 2018103502

Adithya M - 2018103505

Anish Kumar C – 2018103214

Ashwath Narayan K S - 2018103517

ABSTRACT

In the recent years, many researches are done in order to find the concepts within documents. These document units are language's verbs, nouns, adverbs, prepositions etc. that contribute towards building the document. The current application is not limited by picking keywords to understand the document concept but aims to gain a precise understanding of concepts through correlation of words and to classify the documents. In our application we use the Latent Semantic Analysis (LSA) algorithm for movie classification. The training dataset is trained using the algorithm and a matrix is generated. This matrix gives us the correlation of words within documents. By finding the similarity of test dataset with the training dataset, the genre of the test data is classified.

LITERATURE SURVEY

S.NO	PUBLICATION YEAR	AUTHOR	METHODOLOGY	ADVANTAGES	LIMITATION
1	March 15, 2017	Rakshith a GS , Karthik KrishnaMurthi	Word stemming , Stop words removal , data pre – processing	The methodology improves four percentage points for efficiency in pre processing	The method could be further improved by taking more genre information
2	January 5, 2001	Yihong Gong, Xin Liu	Latent semantic Analysis	Text summarization solves the problem of presenting the information needed by a user in a compact form.	LSA has several limitations. The first one is that it does not use the information about word order, syntactic relations and morphologies. This kind of information can be necessary for finding out the meaning of words and texts.

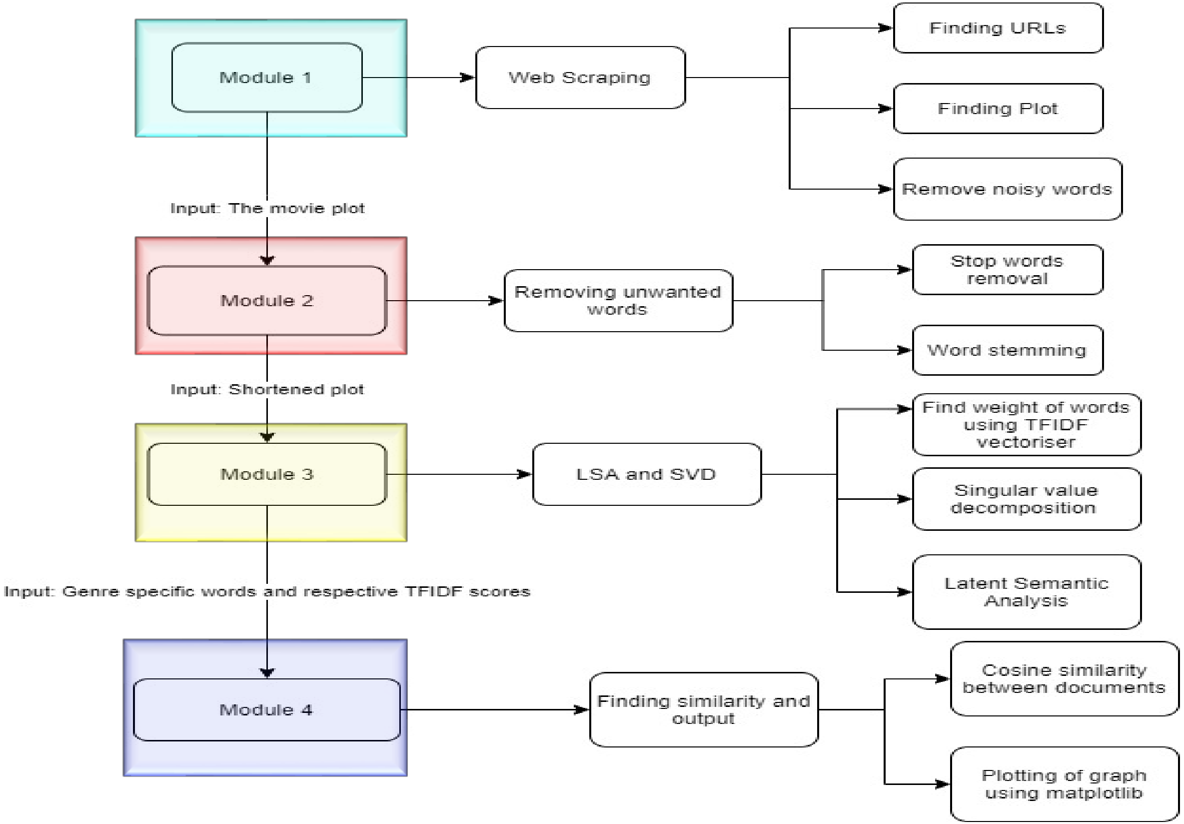
LITERATURE SURVEY

S.NO	PUBLICATION YEAR	AUTHOR	METHODOLOGY	ADVANTAGES	LIMITATION
3	4^{th} IEEE Conference 2016	Alifrna Rizqi lahitani , Adhistya Erna Permanna sari	Cosine similarity	Implemented the waiting of term frequency using TFIDF	Automated scoring depends on many factor and cases such as language etc..

PROBLEM STATEMENT

One of the major problems these days is tagging other movies which fall under similar kinds of storyline or share the same genre. In order to eliminate unnecessary confusion arising due to abundance of movie data online, there must be some software to reduce this conflict of manual tagging of genre. We resort to text summarization. In the current state we are in, huge amounts of time are wasted reading data. So we have implemented a method that takes the key points of the text and displays it. Latent semantic analysis is used here to develop short and concise summaries of their corresponding paragraphs.

SYSTEM ARCHITECTURE



AAKASH K

ASHWATH

ANISH

ADITHYA

Module 1: WEB SCRAPPING

- **Packages used :** urllib , beautiful soup , pandas
- **Input :** csv file with movie title ,release year , genre and other columns.
- **Output :** respective movie plots scraped from Wikipedia

Phases :

1. Scrap the plot
2. Fetch the required plot
3. Removing noisy words and nouns

PHASE 1. Scrap the plot

DESCRIPTION :

From the URL obtained we will have to scrape the plot. For most of the movies the story will be under the plot section. Moreover Wikipedia provides privilege to edit each section. So if we somehow are able to reach the section we can obtain the entire plot of the movie which was uploaded. All these scraping works are done by using Python's Beautiful Soup.

PSEUDO CODE :

```
for i in df.index: //df contains all the movie records ...
    movie_name = df[i]['Title']
    movie_name = movie_name.title()
    movie = ""
    for i in range(len(movie_name)):
        if movie_name[i] == ' ':
            movie += '_'
        else:
            movie += movie_name[i]
    df.loc[i]['Wiki Page'] = movie
```

PHASE 2. Fetch the required plot

Then we will have to fetch the required plot from the edit section.

PSEUDO CODE :

```
url = df.loc[i]['Wiki Page']
        html = urlopen(url)
soup = BeautifulSoup(html, 'html.parser')
url = "https://en.wikipedia.org"+soup.find('a',{'title':'Edit section: Plot'})['href']
```

It has lots and lots of noisy words which are wikipedia related syntaxes , moreover all those words were mostly redirects and also nouns which are nowhere useful in our context , so they'll have to be removed.

PSEUDO CODE :

```
html = urlopen(url)
   soup = BeautifulSoup(html, 'html.parser')
   plot = soup.find('textarea').text
```

PHASE 3. Removing noisy words and nouns

PSEUDO CODE :

```
try:
    i = plot.index('==')
    plot = plot[i+len('=='):]
    i = plot.index('==')
    plot = plot[i+len('=='):]
  except:
    plot = plot
    df.loc[i,'Plot'] = 'a'
def func(pl):
  while(1):
    i = pl.find('[[')
    if(i==-1):
      break
    j = pl.find(']]')
    pl = pl[:i] + pl[j+2:]
  while(1):
    i = pl.find('{{')
    if(i==-1):
      break
    j = pl.find('}}')
    pl = pl[:i] + pl[j+2:]
  while(1):
    i = pl.find('(')
    if(i==-1):
      break
    j = pl.find(')')
    pl = pl[:i] + pl[j+2:]
  p = ''
  for i in pl:
    if(i!='\n'):
      p += i
  return p
```

DESCRIPTION :

The above code will, to some extent, remove the noisy words. Then the output is stored in the 'Plot' column . It is repeated for all the records in the table then it is later stored into the file from which the input was extracted.

Module 2: STOP WORDS REMOVAL AND WORD STEMMING

- **Packages Used :** nltk
- **Input :** Plot of the movie.
- **Output :** Stemmed words.

Phases :

1. Stop words removal
2. Word stemming

Phases : **1.** Stop words removal

DESCRIPTION :

We will have to split the plot into words using nltk's word tokenizer. The first thing we will have to remove is the character names, places in which the story happens , generalising it we need to remove the nouns.

PSEUDO CODE :

```
tagged_sentence = nltk.tag.pos_tag(plot.split())
edited_sentence = [word for word,tag in tagged_sentence if tag != 'NNP' and tag != 'NNPS']
example_sentence = ' '.join(edited_sentence)
```

Phases :

2. Word stemming

DESCRIPTION :

Then the stop words removed list is stemmed using nltk’s Porter Stemmer algorithm. The Porter stemming algorithm or ‘Porter stemmer’ is a process for removing the commoner morphological and in flexional endings from words in English

PSEUDO CODE :

```
punctuations = '''!()-[]{};:'"\,<>./?@#$%^&*_~0123456789'''
no_punct = ""
for char in example_sentence:
   if char not in punctuations:
       no_punct = no_punct + char
   else:
        no_punct += ' '
example_sentence = no_punct
tagged_sentence = nltk.tag.pos_tag(example_sentence.split())
edited_sentence = [word for word,tag in tagged_sentence if tag not in ['NNP','NNPS','NNS','NN']]
print(len(edited_sentence))
example_sentence = ' '.join(edited_sentence)
example_sentence = example_sentence.lower()
stop_words = set(stopwords.words('english'))
word_tokens = word_tokenize(example_sentence)
filtered_sentence = [w for w in word_tokens if not w in stop_words]
```

Module 3: LSA and SVD of stemmed words

Packages Used : sklearn's feature extraction.

Input : Space separated stemmed words.

Output : An Array of genre related words.

Phases :

1. Computing TF and DF
2. Forming TFIDF
3. Singular value decomposition

LSA

Phases : 1. Computing TF and DF

PSEUDO CODE :

```
def computeTF(wordDict, bagOfWords):
  tfDict = {}
  bagOfWordsCount = len(bagOfWords)
  for word, count in wordDict.items():
     tfDict[word] = count /
float(bagOfWordsCount)
  return tfDict
```

where wordDict expects an array of <word , count > pair in a particular document

PSEUDO CODE :

```
def computeIDF(documents):
 N = len(documents)
 idfDict = dict.fromkeys(documents[0].keys(), 0)
 for document in documents:
   for word, val in document.items():
      if val > 0:
        idfDict[word] += 1
 for word, val in idfDict.items():
    idfDict[word] = math.log(N / float(val))
        return idfDict
```

Then tfidf of a document is obtained by multiplying tf and idf values.

Phases : 2. Forming TFIDF

DESCRIPTION :

Based on the tfidf values the matrix will be compressed and the most commonly appearing words will be selected along with their tfidf scores.

PSEUDO CODE :

```
def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf
```

Phases : 3. Singular value decomposition

DESCRIPTION :

Truncated SVD is used for selection of various topics from the document array. Number of topics can be passed as an argument via n_components .

PSEUDO CODE :

```
vectorizer = TfidfVectorizer(min_df = 200,stop_words='english',max_df =
0.9,max_features = 500) bagofwords = vectorizer.fit_transform(plot)
svd = TruncatedSVD(n_components = ncomp)
lsa = svd.fit_transform(bagofwords)
```

Module 4 : Prediction of genres of the movie plot

Packages Used : sklearn's feature extraction, cosine similarity and matplotlib.

Input : Genre specific words and their tfidf scores.

Output : A simple classification graph .

Phases :

1. Cosine similarities method

2. Plotting graph using matplotlib

Phases :

1. Cosine similarities method

DESCRIPTION :

Cosine similarity is a metric used to measure how similar the documents are irrespective of their size. Mathematically, it measures the cosine of the angle between two vectors projected in a multi-dimensional space.

PSEUDO CODE :

```
rat = df2.iloc[df2.shape[0]-len(summa):]
rat = rat.T
rat.index.set_names(['words'],inplace = True)

encoding_matrix = pd.DataFrame(svd.components_,index = topic).T
encoding_matrix['words'] = diction30
diction30 = encoding_matrix.sort_values(by=['Topic 1'],ascending = False)
diction30.index = diction30['words']
topic.remove(top)
diction30 = diction30.drop(topic,axis = 1)
diction30 = diction30.drop(['words'],axis = 1)

comp = pd.concat([rat,diction30], axis=1, join='inner').T
          c = cosine_similarity(comp,comp)
```

Phases : **2.** Plotting graph using matplotlib

DESCRIPTION :

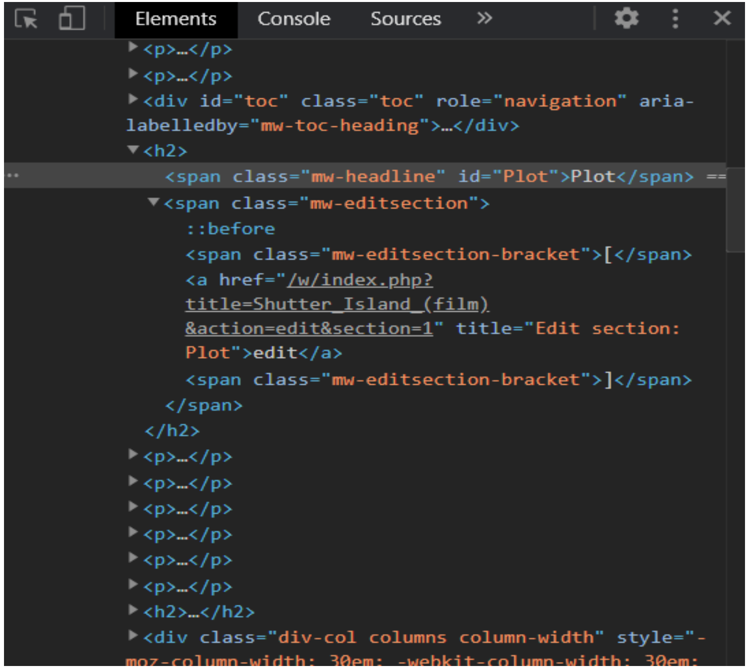
Finally the classification is done via bar graph which shows the diversification of the movie plots among the various genres.

PSEUDO CODE :

```
color = ['#4b0082','#ffd700','#ff4500','#4b0082','#adff2f','#2f4f4f','#00ee4e','#eeee02','#dc143c']
                genre = ans.index.values
genre = [genre[i].title() for i in range(len(genre))]
fig, axs = plt.subplots(math.ceil(ans.shape[1]/2), 2,figsize = (20, 50))
count = 0
for i in ans.columns.values:
    nsix = ans[i] * 100
    axs[(count//2),count%2].bar(genre,nsix.values,color =color, width = 0.4)
    axs[(count//2),count%2].set_title(i)
    count+=1
plt.show()
```

Screen shots of each modules

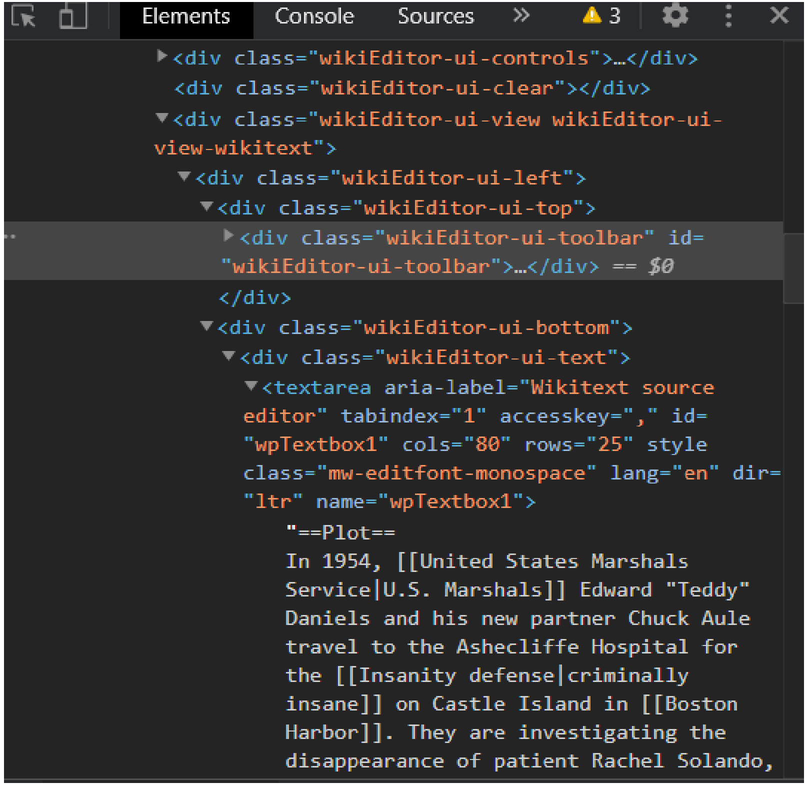
MODULE 1



We can see from the above picture that the edit section is below the Plot's <span>.

We will fetch the edit section's url by finding for the <a>'s href.

MODULE 1



The entire plot is in the <textarea>'s innerText.

MODULE 1

```
Out[18]: 'In 1954, widower U.S. Marshal Edward "Teddy" Daniels and his new partner, Chuck Aule, go on a ferry boat to Shutter Island, th
e home of Ashecliffe Hospital for the criminally insane, to investigate the disappearance of a patient, Rachel Solando  Despite
being kept in a locked cell under constant supervision, she has escaped the hospital and the desolate island.In Rachel\'s room,
Teddy and Chuck discover a code that Teddy breaks. He tells Chuck that he believes the code points to a 67th patient, when reco
rds show only 66. Teddy also reveals that he wants to avenge the death of his wife Dolores, who was murdered two years prior by
a man called Andrew Laeddis, whom he believes is an inmate in Ashecliffe Hospital. The novel is interspersed with graphic descr
iptions of  and , which Teddy helped to liberate. After  hits the island, Teddy and Chuck investigate Ward C, where Teddy belie
ves government experiments with psychotropic drugs are being conducted. While separated from Chuck for a short while in Ward C,
Teddy meets a patient called George Noyce, who tells him that everything is an elaborate game designed for him, and that Chuck
is not to be trusted.As Teddy and Chuck return to the main hospital area, they are separated. Teddy discovers a woman who says
she is the real Rachel Solando. She tells him she was actually a psychiatrist at Ashecliffe, and when she discovered the illega
l experiments being run by them, she was incarcerated as a patient. She escaped and has been hiding in different places on the
island. She warns him about the other residents of the island, telling him to take care with the food, medication and cigarette
s, which have been laced with psychotropic drugs. When Teddy returns to the hospital, he can\'t find Chuck and is told he had n
o partner. He escapes and tries to rescue Chuck at the lighthouse, where he believes the experiments take place. He reaches the
top of the lighthouse and finds only hospital administrator Dr. Cawley seated at a desk. Cawley tells Teddy that he himself is
in fact Andrew Laeddis and that he has been a patient at Shutter Island for two years for murdering his wife, Dolores Chanal  a
fter she murdered their three children.Andrew/Teddy refuses to believe this and takes extreme measures to disprove it, grabbing
what he thinks is his gun and tries to shoot Dr. Cawley; but the weapon is a toy water pistol. Chuck then enters, revealing tha
t he is actually Andrew\'s psychiatrist, Dr. Lester Sheehan. He is told that Dr. Cawley and Chuck/Sheehan have devised this tre
atment to allow him to live out his elaborate fantasy, in order to confront the truth, or else undergo a radical  treatment. Te
ddy/Andrew accepts that he killed his wife and his service as a US Marshal was a long time ago.The ending of the novel has Tedd
y receive a lobotomy in order to avoid living with the knowledge that his wife murdered their children and he is her murderer.'
```

Movie Plot scrapped from wikipedia for the movie Shutter Island

MODULE 2

```
In [25]: filtered_sentence

Out[25]: ['widower',
          'new',
          'go',
          'criminally',
          'investigate',
          'despite',
          'kept',
          'locked',
          'constant',
          'escaped',
          'discover',
          'breaks',
          'tells',
          'believes',
          'show',
          'also',
          'reveals',
          'wants',
          'avenge',
          'murdered',
          'two',
          'prior',
          'called',
          'believes',
          'interspersed',
          'graphic',
          'helped',
          'liberate',
          'investigate',
          'believes',
          'conducted',
          'separated',
          'short',
          'meets',
          'called',
          'tells',
          'elaborate',
          'designed',
          'trusted',
          'return',
          'main',
          'separated',
```

filt = [ps.stem(filtered_sentence[i]) for i in range(len(filtered_sentence))]

MODULE 3

jupyter Untitled3 Last Checkpoint: Last Wednesday at 10:14 PM (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Trusted | Python 3

Run Code

Out[214]:

words	Topic 1	Topic 2	Topic 3	Topic 4	Topic 5	Topic 6	Topic 7	Topic 8	Topic 9	Topic 10	Topic 11	Topic 12	Topic 13	Topic 14
kill	0.513598	0.110373	0.213720	0.591858	-0.029882	0.331752	-0.125819	0.118201	0.277197	-0.036523	0.174364	-0.007483	0.134970	-0.074727
murder	0.476790	0.555194	-0.073404	-0.261924	0.055811	-0.170873	0.118972	-0.047195	-0.125823	-0.026153	-0.063135	0.005841	-0.072633	0.032118
reveal	0.388038	0.326916	-0.058295	-0.147050	0.008132	-0.121729	-0.029285	-0.060753	-0.080972	0.051349	-0.079851	-0.041678	-0.026517	0.049331
leav	0.164421	-0.220969	-0.170359	0.006011	-0.332976	-0.380526	0.059084	-0.188661	0.091676	-0.390579	-0.116657	-0.171694	0.335961	0.045233
tell	0.163214	-0.215817	-0.326595	0.267995	0.113559	-0.321741	0.041978	-0.232307	0.206852	0.004910	0.005396	-0.075309	-0.300626	-0.293012
make	0.142414	-0.168749	0.010674	-0.352508	0.105530	0.399555	0.401356	-0.408512	0.428792	0.064018	0.015575	0.115513	-0.174529	-0.027626
use	0.136462	-0.165229	0.142704	0.019033	-0.238791	0.313802	0.030195	-0.101987	-0.295446	-0.160967	-0.416764	-0.136346	-0.273124	0.015229
escap	0.135882	-0.229348	0.767943	-0.052887	0.231585	-0.464284	0.040367	-0.006211	0.042892	-0.070712	-0.035255	0.079866	-0.106123	0.015206
tri	0.123040	-0.161961	0.066624	-0.011612	0.138631	0.222601	-0.177891	-0.256603	-0.076615	-0.010656	0.062982	-0.274223	0.259929	0.326235
come	0.122808	-0.141618	-0.181818	-0.098369	0.459072	0.108759	-0.221583	0.128355	-0.133353	-0.255197	-0.207905	0.365366	0.167808	-0.163029
goe	0.122126	-0.180753	-0.194813	-0.029655	0.371294	0.036153	0.138739	0.264220	-0.305582	-0.173461	0.376779	-0.216825	-0.232380	-0.082056
later	0.118531	-0.153936	0.040754	-0.067584	-0.125839	0.075698	-0.139002	-0.009043	-0.255652	0.114932	-0.097285	0.197006	0.090142	-0.061813
begin	0.116965	-0.109352	-0.068407	-0.214608	-0.219996	0.019700	-0.154435	0.259202	0.320478	-0.277341	0.152225	0.296006	-0.054629	-0.157261
ask	0.116595	-0.159689	-0.179501	0.191871	-0.030346	-0.138306	-0.216903	-0.077699	-0.032334	0.253194	0.029361	0.204437	-0.405104	0.362243
becom	0.115621	-0.125759	0.038220	-0.432613	-0.236286	0.008915	-0.443584	0.123509	0.128552	0.007228	0.315411	-0.283533	-0.062693	-0.054257
away	0.111081	-0.138257	0.016162	0.056232	-0.085322	0.022131	0.161862	0.014088	-0.161530	-0.228590	0.007070	0.083748	-0.160443	0.042796
follow	0.110262	-0.142731	-0.019894	-0.026547	-0.216658	0.024458	0.494003	0.210220	-0.162678	0.008929	0.046029	-0.009711	0.240177	-0.154369
discov	0.108388	-0.097415	-0.018960	-0.051698	-0.301348	-0.037128	-0.001700	0.003711	-0.188578	0.280373	0.133137	0.202688	-0.100653	0.029313
die	0.107056	-0.116075	0.018150	0.081837	-0.153725	0.007024	-0.012017	0.109015	-0.086688	0.284095	-0.128720	-0.003145	0.001760	-0.284925
dead	0.106350	-0.124259	-0.000219	0.007351	0.039397	-0.086291	0.269031	0.009537	-0.107338	0.281377	0.438945	0.117404	0.266355	0.161133
meet	0.105343	-0.139929	-0.152383	-0.037878	0.063668	-0.039617	0.161202	0.572999	0.321960	0.121043	-0.372610	-0.099888	0.001927	0.473886

These are the words selected for thriller genre and their corresponding scores in various topics among this one of the topics will be selected by finding the maximum sum obtained by the test plots which was separated earlier.

MODULE 3

```
In [223]: cons

Out[223]: Topic 1     109.699096
          Topic 2     -19.344380
          Topic 3      -3.944621
          Topic 4      -3.214067
          Topic 5       0.963171
          Topic 6      -0.400785
          Topic 7       3.730655
          Topic 8      -3.020649
          Topic 9       0.003026
          Topic 10     -0.247261
          Topic 11      0.474438
          Topic 12      2.301618
          Topic 13      1.415471
          Topic 14      0.032225
          Topic 15     -1.668353
          Name: sum, dtype: float64
```

Sum obtained by the respective topics

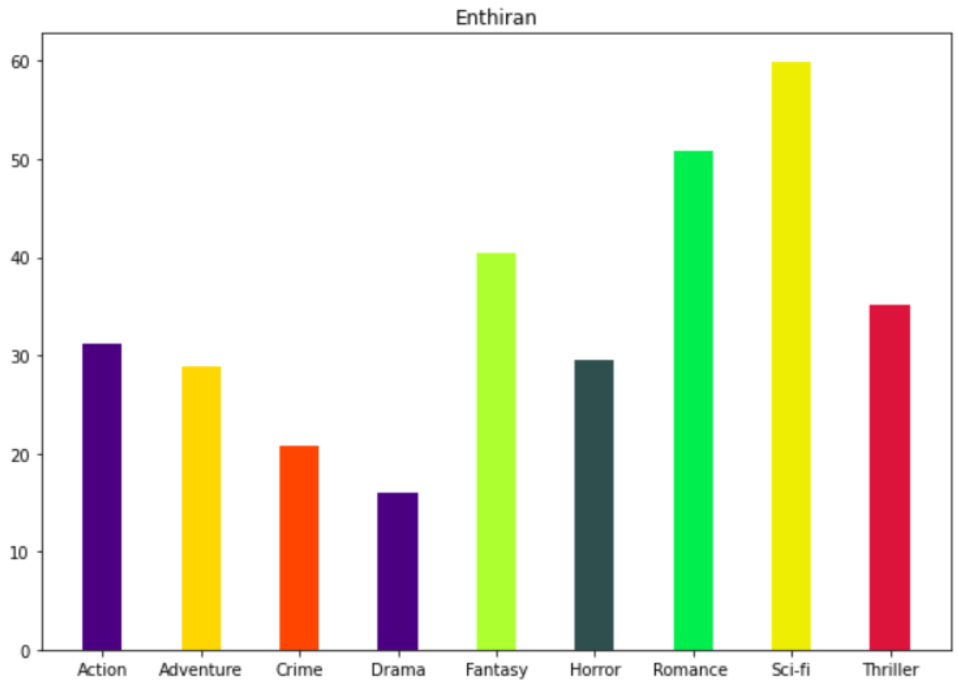
MODULE 4

Out[231]:

words	enthiran	topic 1
anoth	0.000000	0.089511
arriv	0.000000	0.081676
away	0.000000	0.100021
becom	0.180702	0.104046
begin	0.000000	0.102169
believ	0.000000	0.080221
dead	0.000000	0.095349
decid	0.000000	0.070258
die	0.000000	0.098429
discov	0.000000	0.096369
end	0.000000	0.071712
escap	0.000000	0.128930
final	0.000000	0.081127
follow	0.187472	0.099583
goe	0.000000	0.112872
help	0.402415	0.083206
kill	0.215376	0.491658
know	0.000000	0.085269
later	0.000000	0.110204
lead	0.221706	0.069354
learn	0.000000	0.073387
leav	0.000000	0.148066

Tfidf scores of thriller dataset and input plot for which cosine similarity needs to be calculated.

MODULE 4



An example classification graph for a science fiction movie

REFERENCES

[1] Dr. Edel Garcia “Latent Semantic Indexing (LSI) A Fast Track Tutorial”, FirstPublished on September 21, 2006 Last Update: October 21, 2006

[2] Kanejiya, A. Kumar and S. Prasad, “Automatic Evaluation of StudentsAnswers using Syntactically Enhanced LSA”, Workshop on BuildingEducational Applications using NLP, 53–60, 2003.

[3] Pimwadee Chaovalit, Lina Zhou “Movie Review Mining: a Comparisonbetween Supervised and Unsupervised Classification Approaches”,Proceedings of the 38th Hawaii International Conference on System Sciences– 2005.

[4] Karthik Krishnamurthi1, Vijayapal Reddy Panuganti2, Vishnu VardhanBulusu3, Influence of Supplementary Information on the Semantic Structureof Documents, International Journal of Advanced Research in Computer andCommunication Engineering Vol. 4, Issue 7, July 2015.

THANK YOU ALL