Movie Genre Prediction(MGP) using NLP

Hemansh Anand

Notebook 1: EDA and Encoding the Dataset

The main purpose of this notebook is to analyse and visualise the dataset. The irrelevant features(columns) in the dataset will be dropped and only the relevant fautures will be retained. We will then encode the genres into the dataset and make a new dataset ready for Pre Processing in the next notebook.

This notebook accomplishes two primary tasks:

1. Analyse and Visualise the Dataset
2. Encoding the Genres into the Datset

Problem Statement and Motivation

The goal of this project is to build a machine mearning model in order to predict movie genres based on their plot descriptions. These are the sub goals of this project:

- Apply TF-IDF transformation to movie plot.
- Use classic machine-learning techniques to classify movies as one or more genres using the transformed plots.

Import the Libraries and Dataset

In [ ]:

```
import pandas as pd
import numpy as np
import json
import nltk
import re
import csv
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

%matplotlib inline
```

In [ ]:

```
df = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/movie.metadata.tsv',sep = '\t',
header = None)
```

In [ ]:

```
df.head()
```

Out[ ]:

	0	1	2	3	4	5	6	7	8
0	975900	/m/03vyhn	Ghosts of Mars	2001-08-24	14010832.0	98.0	{"/m/02h40lc": "English Language"}	{"/m/09c7w0": "United States of America"}	{"/m/01jfsb": "Thriller", "/m/06n90": "Science Fiction", "/m/03npn": "Horror", "/m/03k9fj": "Adventure", "/m/0fdjb": "Supernatural", "/m/02kdv5l": "Action", "/m/09zvmj": "Space western"}
1	3196793	/m/08yl5d	Getting Away with Murder: The JonBenét Ramsey Mystery	2000-02-16	NaN	95.0	{"/m/02h40lc": "English Language"}	{"/m/09c7w0": "United States of America"}	{"/m/02n4kr": "Mystery", "/m/03bxz7": "Biographical film", "/m/07s9rl0": "Drama", "/m/0hj3n01": "Crime Drama"}

	0	1	2	3	4	5	6	7	8
2	28463795	/m/0crgdbh	Brun bitter	1988	NaN	83.0	{"/m/05f_3": "Norwegian Language"}	{"/m/05b4w": "Norway"}	{"/m/0lsxr": "Crime Fiction", "/m/07s9rl0": "Drama"}
3	9363483	/m/0285_cd	White Of The Eye	1987	NaN	110.0	{"/m/02h40lc": "English Language"}	{"/m/07ssc": "United Kingdom"}	{"/m/01jfsb": "Thriller", "/m/0glj9q": "Erotic thriller", "/m/09blyk": "Psychological thriller"}
			A Woman in				{"/m/04306rv":	{"/m/0345h":	

Renaming Columns

In [ ]:
```
df.columns = ["movie_id",1,"movie_name",3,4,5,6,7,"genre"]
```

In [ ]:
```
plots = []

with open("/content/drive/MyDrive/Colab Files/MGP/Datasets/plot_summaries.txt", 'r') as plot_txt:
      reader = csv.reader(plot_txt, dialect='excel-tab')
      for row in tqdm(reader):
           plots.append(row)
```

```
42303it [00:00, 44296.35it/s]
```

In [ ]:

```
movie_id = []
plot = []

# extract movie Ids and plot summaries
for i in tqdm(plots):
  movie_id.append(i[0])
  plot.append(i[1])

# create dataframe
movies = pd.DataFrame({'movie_id': movie_id, 'plot': plot})
```

```
100%|██████████| 42303/42303 [00:00<00:00, 1033761.03it/s]
```

In [ ]:

```
movies.head()
```

Out[ ]:

	movie_id	plot
0	23890098	Shlykov, a hard-working taxi driver and Lyosha, a saxophonist, develop a bizarre love-hate relationship, and despite their prejudices, realize they aren't so different after all.
1	31186339	The nation of Panem consists of a wealthy Capitol and twelve poorer districts. As punishment for a past rebellion, each district must provide a boy and girl between the ages of 12 and 18 selected by lottery for the annual Hunger Games. The tributes must fight to the death in an arena; the sole...
2	20663735	Poovalli Induchoodan is sentenced for six years prison life for murdering his classmate. Induchoodan, the only son of Justice Maranchery Karunakara Menon was framed in the case by Manapally Madhavan Nambiar and his crony DYSP Sankaranarayanan to take revenge on idealist judge Menon who had e...
3	2231378	The Lemon Drop Kid , a New York City swindler, is illegally touting horses at a Florida racetrack. After several successful hustles, the Kid comes across a beautiful, but gullible, woman intending to bet a lot of money. The Kid convinces her to switch her bet, employing a prefabricated con. Unfo...
4	595909	Seventh-day Adventist Church pastor Michael Chamberlain, his wife Lindy, their two sons, and their nine-week-old daughter Azaria are on a camping holiday in the Outback. With the baby sleeping in their tent, the family is enjoying a barbecue with their fellow campers when a cry is heard. Lindy r...

Changing the data type of movie id and merging the dataframes

In [ ]:

```
# change datatype of 'movie_id'
df['movie_id'] = df['movie_id'].astype(str)


# merge meta with movies
movies = pd.merge(movies, df[['movie_id', 'movie_name', 'genre']], on = 'movie_id')


movies.head()
```

Out[ ]:

	movie_id	plot	movie_name	genre
0	23890098	Shlykov, a hard-working taxi driver and Lyosha, a saxophonist, develop a bizarre love-hate relationship, and despite their prejudices, realize they aren't so different after all.	Taxi Blues	{"/m/07s9rl0": "Drama", "/m/03q4nz": "World cinema"}
1	31186339	The nation of Panem consists of a wealthy Capitol and twelve poorer districts. As punishment for a past rebellion, each district must provide a boy and girl between the ages of 12 and 18 selected by lottery for the annual Hunger Games. The tributes must fight to the death in an arena; the sole...	The Hunger Games	{"/m/03btsm8": "Action/Adventure", "/m/06n90": "Science Fiction", "/m/02kdv5l": "Action", "/m/07s9rl0": "Drama"}
2	20663735	Poovalli Induchoodan is sentenced for six years prison life for murdering his classmate. Induchoodan, the only son of Justice Maranchery Karunakara Menon was framed in the case by Manapally Madhavan Nambiar and his crony DYSP Sankaranarayanan to take revenge on idealist judge Menon who had e...	Narasimham	{"/m/04t36": "Musical", "/m/02kdv5l": "Action", "/m/07s9rl0": "Drama", "/m/01chg": "Bollywood"}
3	2231378	The Lemon Drop Kid , a New York City swindler, is illegally touting horses at a Florida racetrack. After several successful hustles, the Kid comes across a beautiful, but gullible, woman intending to bet a lot of money. The Kid convinces her to switch her bet, employing a prefabricated con. Unfo...	The Lemon Drop Kid	{"/m/06qm3": "Screwball comedy", "/m/01z4y": "Comedy"}
4	595909	Seventh-day Adventist Church pastor Michael Chamberlain, his wife Lindy, their two sons, and their nine-week-old daughter Azaria are on a camping holiday in the Outback. With the baby sleeping in their tent, the family is enjoying a barbecue with their fellow campers when a cry is heard. Lindy r...	A Cry in the Dark	{"/m/0lsxr": "Crime Fiction", "/m/07s9rl0": "Drama", "/m/01f9r0": "Docudrama", "/m/03q4nz": "World cinema", "/m/05bh16v": "Courtroom Drama"}

In [ ]:

```
movies['genre'][0]
```

Out[ ]:

```
'{"/m/07s9rl0": "Drama", "/m/03q4nz": "World cinema"}'
```

In [ ]:
```
type(json.loads(movies['genre'][0]))
```

Out[ ]:
```
dict
```

In [ ]:
```
json.loads(movies['genre'][0]).values()
```

Out[ ]:
```
dict_values(['Drama', 'World cinema'])
```

In [ ]:
```
# an empty list
genres = []

# extract genres
for i in movies['genre']:
  genres.append(list(json.loads(i).values()))

# add to 'movies' dataframe
movies['genre_new'] = genres
```

In [ ]:
```
del movies['genre']
```

In [ ]:
```
movies.head()
```

Out[ ]:

	movie_id	plot	movie_name	genre_new
0	23890098	Shlykov, a hard-working taxi driver and Lyosha, a saxophonist, develop a bizarre love-hate relationship, and despite their prejudices, realize they aren't so different after all.	Taxi Blues	[Drama, World cinema]
1	31186339	The nation of Panem consists of a wealthy Capitol and twelve poorer districts. As punishment for a past rebellion, each district must provide a boy and girl between the ages of 12 and 18 selected by lottery for the annual Hunger Games. The tributes must fight to the death in an arena; the sole...	The Hunger Games	[Action/Adventure, Science Fiction, Action, Drama]

	movie_id	plot	movie_name	genre_new
2	20663735	Poovalli Induchoodan is sentenced for six years prison life for murdering his classmate. Induchoodan, the only son of Justice Maranchery Karunakara Menon was framed in the case by Manapally Madhavan Nambiar and his crony DYSP Sankaranarayanan to take revenge on idealist judge Menon who had e...	Narasimham	[Musical, Action, Drama, Bollywood]
3	2231378	The Lemon Drop Kid , a New York City swindler, is illegally touting horses at a Florida racetrack. After several successful hustles, the Kid comes across a beautiful, but gullible, woman intending to bet a lot of money. The Kid convinces her to switch her bet, employing a prefabricated con. Unfo...	The Lemon Drop Kid	[Screwball comedy, Comedy]

Remove samples with zero genre tags

In [ ]:
```
movies_new = movies[~(movies['genre_new'].str.len() == 0)]
```

In [ ]:
```
movies_new.shape, movies.shape
```

Out[ ]:
```
((41793, 4), (42204, 4))
```

Removed 411 samples with no genre tags

Let's find the number of movie genres which have been covered in this dataset

In [ ]:
```
all_genres = sum(genres,[])
len(set(all_genres))
```

Out[ ]:
```
363
```

There are 363 Genres in this dataset. Let's have a closer look at the genres

Using FreqDist() from nltk libarary, let's create a dictionary of genres and their count accross our dataset.

In [ ]:
```
all_genres_dict = nltk.FreqDist(all_genres)
```

Convert it to a dataframe

In [ ]:
```
all_genres_df = pd.DataFrame({'Genre': list(all_genres_dict.keys()), 'Count':
list(all_genres_dict.values())})
```

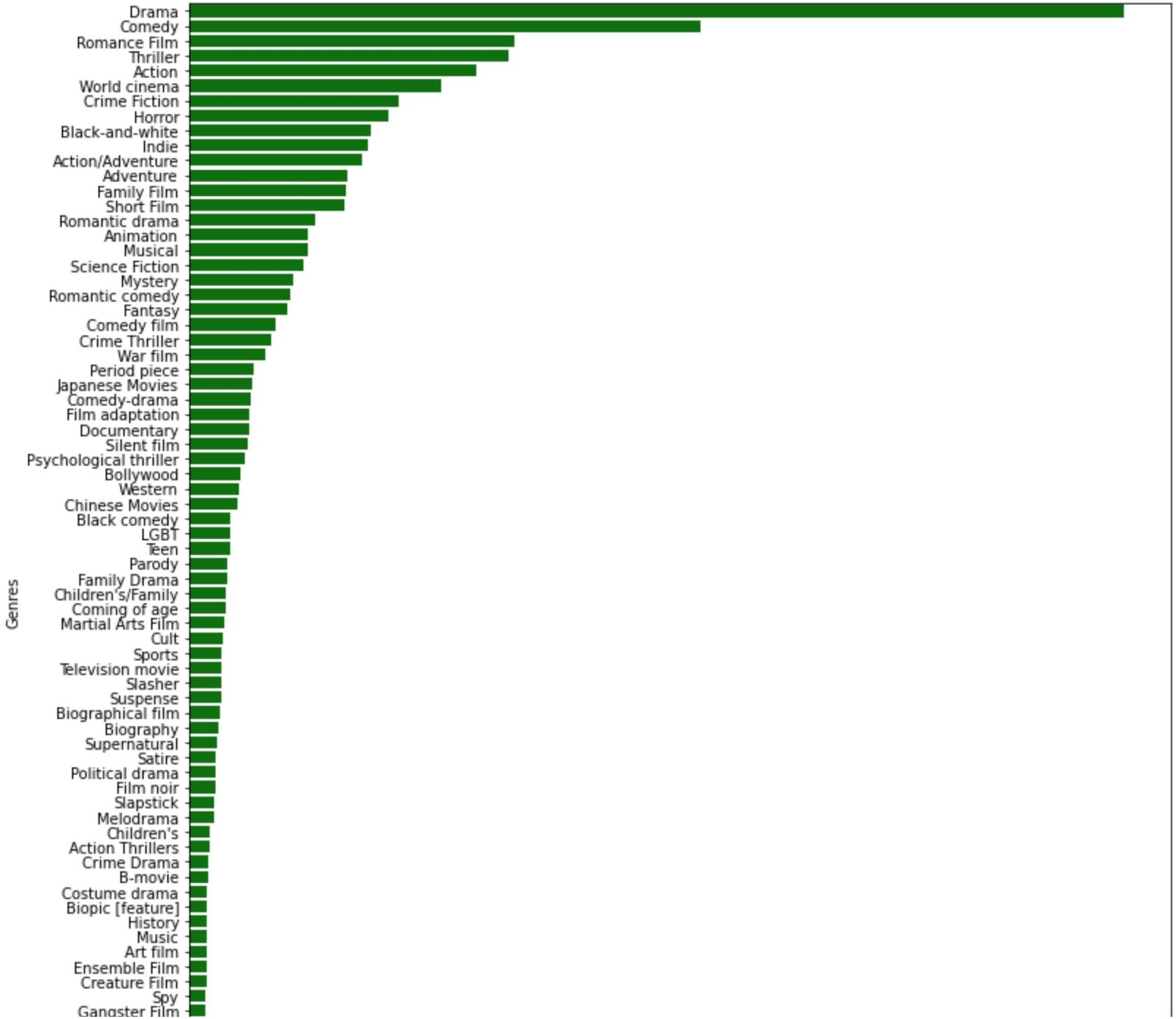
In [ ]:
```
all_genres_df
```

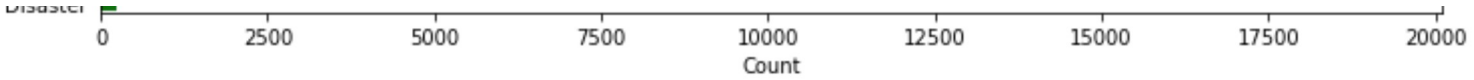
Out[ ]:

	Genre	Count
0	Drama	19134
1	World cinema	5153
2	Action/Adventure	3553
3	Science Fiction	2339
4	Action	5868
...	...	...
358	Statutory rape	1
359	Silhouette animation	1
360	Children's Issues	1
361	Homoeroticism	1
362	Neorealism	1

363 rows × 2 columns

In [ ]:
```
gen = all_genres_df.nlargest(columns="Count", n = 80)
plt.figure(figsize=(12,15))
ax = sns.barplot(data=gen, x= "Count", y = "Genre",color='green')
ax.set(ylabel = 'Genres',xlabel = 'Count')
plt.show()
```


Genres
Drama
Comedy
Romance Film
Thriller
Action
World cinema
Crime Fiction
Horror
Black-and-white
Indie
Action/Adventure
Adventure
Family Film
Short Film
Romantic drama
Animation
Musical
Science Fiction
Mystery
Romantic comedy
Fantasy
Comedy film
Crime Thriller
War film
Period piece
Japanese Movies
Comedy-drama
Film adaptation
Documentary
Silent film
Psychological thriller
Bollywood
Western
Chinese Movies
Black comedy
LGBT
Teen
Parody
Family Drama
Children's/Family
Coming of age
Martial Arts Film
Cult
Sports
Television movie
Slasher
Suspense
Biographical film
Biography
Supernatural
Satire
Political drama
Film noir
Slapstick
Melodrama
Children's
Action Thrillers
Crime Drama
B-movie
Costume drama
Biopic [feature]
History
Music
Art film
Ensemble Film
Creature Film
Spy
Gangster Film



These are just the top 80 Genres displayed.

It is difficult to build a prediction model with good accuracy because:

- There are too many classes
- Many of the sub genre classes have few observation as compared to the main genres

Solution to this problem?

We need to reduce the number of genres in this dataset. This can be achieved by:

- Absorbing or Clubbing sub genres to a main group of genres
- Removing redundant genres

But this solution is time consuming and this time can be better invested in trying out different experiment setups rather than cleaning the data. So, I decided to use another dataset which has just 25 Genres.

I had proposed the idea to the group to change the dataset but 2 group members had already made progress in their experiments so the group declined to change the dataset.

Justification to use different dataset:

I decided to go forward with this new dataset as the data contains more or less the same movies but it's a better and much cleaner data. The end result will be same, i.e. Movie Genre Prediction and if the model gets trained better because of the quality of this data it will be beneficial to the group in the deployment stage. As decided in the group, I will still be trying out the same setups that I have been assigned to me.

New Dataset

In [ ]:
```
df = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/imdb_dataset.tsv', sep='\t',
index_col=0)
df.head(5)
```

Out[ ]:

	tconst	title	release_year	release_date	MPAA	genre	runtime	poster_url	plot_short	plot_long	imdb_rating	num_imdb_vo
0	tt0010323	Das Cabinet des Dr. Caligari	1920	27 February 1920	Not Rated	['Fantasy', 'Horror', 'Mystery', 'Thriller']	67	https://m.media-amazon.com/images/M/MV5BNWJiNG...	Hypnotist Dr. Caligari uses a somnambulist, Ce...	Francis, a young man, recalls in his memory th...	8.1	52
1	tt0011237	Der Golem, wie er in die Welt kam	1920	19 June 1921	Unrated	['Fantasy', 'Horror']	91	https://m.media-amazon.com/images/M/MV5BMTQ5MT...	In 16th-century Prague, a rabbi creates the Go...	In 16th-century Prague, a rabbi creates the Go...	7.2	6
2	tt0011841	Way Down East	1920	3 September 1920	Passed	['Drama', 'Romance']	145	https://m.media-amazon.com/images/M/MV5BMjI5OT...	A naive country girl is tricked into a sham ma...	The callous rich, portrayed by Lennox, think o...	7.4	4
3	tt0011130	Dr. Jekyll and Mr. Hyde	1920	28 September 1920	Unrated	['Drama', 'Horror', 'Sci-Fi']	82	https://m.media-amazon.com/images/M/MV5BMzE5Yj...	Dr. Henry Jekyll experiments with scientific m...	Based on the Robert Louis Stevenson story: Doc...	7.0	4

tconst	title	release_year	release_date	MPAA	genre	runtime	poster_url	plot_short	plot_long	imdb_rating	num_imdb_vo
									Southern		

This dataset has many columns which we don't require in our model building, hence it's better to drop down these columns.

In [ ]:
```
df =
df.drop(['tconst','title','release_year','release_date','MPAA','runtime','poster_url','imdb_rating',
                'num_imdb_votes','plot_full','metacritic','num_user_reviews','num_critic_reviews'],
axis=1)
df.head()
```

Out[ ]:

	genre	plot_short	plot_long
0	['Fantasy', 'Horror', 'Mystery', 'Thriller']	Hypnotist Dr. Caligari uses a somnambulist, Ce...	Francis, a young man, recalls in his memory th...
1	['Fantasy', 'Horror']	In 16th-century Prague, a rabbi creates the Go...	In 16th-century Prague, a rabbi creates the Go...
2	['Drama', 'Romance']	A naive country girl is tricked into a sham ma...	The callous rich, portrayed by Lennox, think o...
3	['Drama', 'Horror', 'Sci-Fi']	Dr. Henry Jekyll experiments with scientific m...	Based on the Robert Louis Stevenson story: Doc...
4	['Drama', 'Romance']	Abandoned by her fiancé, an educated black wom...	Southern negro Sylvia Landry visits her cousin...

Let's check for null values if any

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
genre          0
plot_short    37
plot_long     37
dtype: int64
```

In [ ]:
```
df = df.dropna(axis=0, subset=['plot_short'])
```

In [ ]:
```
df.isna().sum()
```

Out[ ]:
```
genre         0
```

```
plot_short    0
plot_long     0
dtype: int64
```

There are no null values in the data now.

Let's find out the number of genres this dataset contains

In [ ]:
```
total_genres = []


for row in df['genre']:
    i = row.strip('[]').replace("'",'').split(', ')
    for gen in i:
        total_genres.append(gen)


total_genres = list(set(total_genres))
```

In [ ]:
```
len(total_genres)
```

Out[ ]: 25

Let's sort the genres alphabetically and view the list

In [ ]:
```
total_genres.sort()
total_genres = [i.lower() for i in total_genres]
print(total_genres)
```

```
['action', 'adult', 'adventure', 'animation', 'biography', 'comedy', 'crime', 'documentary', 'drama', 'family', 'fant
asy', 'film-noir', 'game-show', 'history', 'horror', 'music', 'musical', 'mystery', 'news', 'romance', 'sci-fi', 'spo
rt', 'thriller', 'war', 'western']
```

It's time to encode these genres.

To do this task, I could have used the MultiLabelBinarizer() but I felt that simply iterating over the data was an efficient solution.

First, I have created 25 new columns for the 25 genres in this dataset.

In [ ]:

```
for gen in total_genres:
    df.loc[:, gen] = 0
```

In [ ]:

```
df.head()
```

Out[ ]:

	genre	plot_short	plot_long	action	adult	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir
0	['Fantasy', 'Horror', 'Mystery', 'Thriller']	Hypnotist Dr. Caligari uses a somnambulist, Ce...	Francis, a young man, recalls in his memory th...	0	0	0	0	0	0	0	0	0	0	0	0
1	['Fantasy', 'Horror']	In 16th-century Prague, a rabbi creates the Go...	In 16th-century Prague, a rabbi creates the Go...	0	0	0	0	0	0	0	0	0	0	0	0
2	['Drama', 'Romance']	A naive country girl is tricked into a sham ma...	The callous rich, portrayed by Lennox, think o...	0	0	0	0	0	0	0	0	0	0	0	0
3	['Drama', 'Horror', 'Sci-Fi']	Dr. Henry Jekyll experiments with scientific m...	Based on the Robert Louis Stevenson story: Doc...	0	0	0	0	0	0	0	0	0	0	0	0
4	['Drama', 'Romance']	Abandoned by her fiancé,	Southern negro	0	0	0	0	0	0	0	0	0	0	0	0

	genre	plot_short	plot_long	action	adult	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir
		an educated	Sylvia Landry												

Columns have been created for the 25 genres in our dataset. Now I will encode the genre columns.

In [ ]:
```
for i in df.index:
    for g in total_genres:
        df.loc[ df.index == i, g ] = int(g in df['genre'][i].lower())
    if i % 100 == 0:
        print(i, end='\r')
```

In [ ]:
```
df.head()
```

Out[ ]:

	genre	plot_short	plot_long	action	adult	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir
0	['Fantasy', 'Horror', 'Mystery', 'Thriller']	Hypnotist Dr. Caligari uses a somnambulist, Ce...	Francis, a young man, recalls in his memory th...	0	0	0	0	0	0	0	0	0	0	1	0
1	['Fantasy', 'Horror']	In 16th-century Prague, a rabbi creates the Go...	In 16th-century Prague, a rabbi creates the Go...	0	0	0	0	0	0	0	0	0	0	1	0
2	['Drama', 'Romance']	A naive country girl is tricked into a sham ma...	The callous rich, portrayed by	0	0	0	0	0	0	0	0	1	0	0	0

	genre	plot_short	plot_long	action	adult	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir
			Lennox, think o...												
3	['Drama', 'Horror', 'Sci-Fi']	Dr. Henry Jekyll experiments with scientific m...	Based on the Robert Louis Stevenson story: Doc...	0	0	0	0	0	0	0	0	1	0	0	0
4	['Drama', 'Romance']	Abandoned by her fiancé, an educated	Southern negro Sylvia Landry	0	0	0	0	0	0	0	0	1	0	0	0

In [ ]:

```
genre_gin = []
genre = []

for g in total_genres:
    genre.append(g)
    genre_gin.append(df[g].value_counts()[1])

genre_dict = dict(zip(genre, genre_gin))

genre_df = pd.DataFrame.from_dict(genre_dict,orient='index')
genre_df = genre_df.reset_index()
genre_df.columns = ['Genre','Count']
genre_df = genre_df.sort_values('Count', ascending=False)
genre_df
```

Out[ ]:

	Genre	Count

	Genre	Count
8	drama	16750
5	comedy	10564
22	thriller	7515
19	romance	6732
0	action	5490
6	crime	4919
14	horror	4033
2	adventure	3339
17	mystery	2776
20	sci-fi	2477
10	fantasy	2354
15	music	1869
9	family	1809
4	biography	1599
23	war	1493
13	history	1382
7	documentary	1333
3	animation	965
16	musical	874
21	sport	794
24	western	676
11	film-noir	311
18	news	41

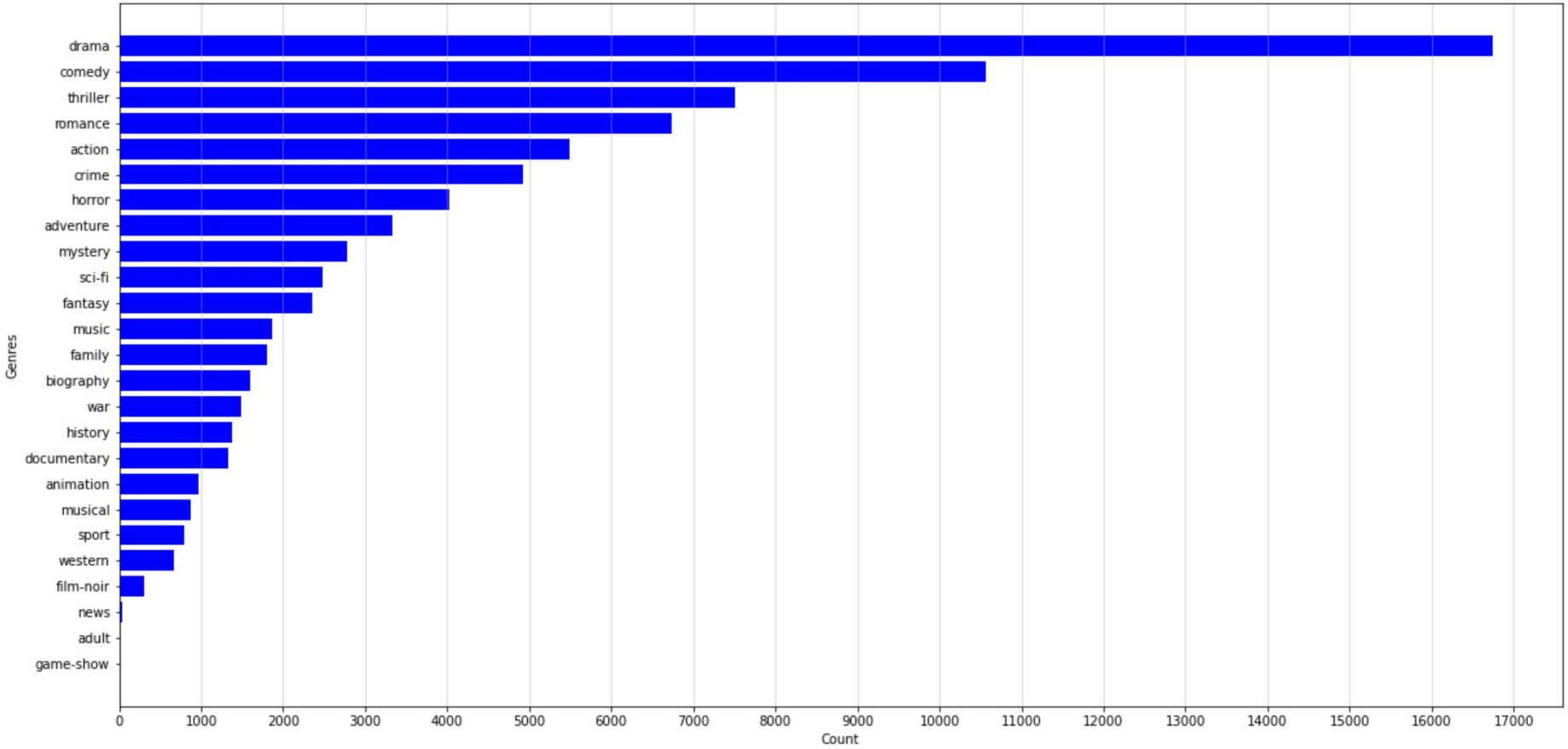
Let's visualise this data and find out the outliers

In [ ]:

```
genre_df = genre_df.sort_values('Count', ascending=True)
plt.figure(figsize=(20,10))
plt.barh(genre_df['Genre'], genre_df['Count'], color='blue')

plt.ylabel('Genres')
plt.xlabel('Count')

plt.grid(axis='x', alpha=0.5)
plt.show()
```



After looking at this plot, it can be noticed that the count of **News, Adult and Game-Show** is negligible with respect to other genres and can be dropped.

Let's drop these genres and make a new dataframe which will then be exported.

In [ ]:

```
df_2 = df
df_2 = df_2.drop(['news','adult','game-show','genre'], axis=1)
df_2
```

Out[ ]:

	plot_short	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horrc

	plot_short	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horrc
0	Hypnotist Dr. Caligari uses a somnambulist, Ce...	Francis, a young man, recalls in his memory th...	0	0	0	0	0	0	0	0	0	1	0	0	
1	In 16th-century Prague, a rabbi creates the Go...	In 16th-century Prague, a rabbi creates the Go...	0	0	0	0	0	0	0	0	0	1	0	0	
2	A naive country girl is tricked into a sham ma...	The callous rich, portrayed by Lennox, think o...	0	0	0	0	0	0	0	1	0	0	0	0	
3	Dr. Henry Jekyll experiments with scientific m...	Based on the Robert Louis Stevenson story: Doc...	0	0	0	0	0	0	0	1	0	0	0	0	
4	Abandoned by her fiancé, an educated black wom...	Southern negro Sylvia Landry visits her cousin...	0	0	0	0	0	0	0	1	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
30077	The story is about friendship and	The story is about friendship	1	0	0	0	1	0	0	0	0	0	0	0	

	plot_short	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horrc
	family witch...	and family witch...													
30078	Feature adaptation of Rapman's YouTube series ...	Blue Story is a tragic tale of a friendship be...	0	0	0	0	0	1	0	1	0	0	0	0	
30079	A man returns home to Atlanta to help his brot...	A good man (Tip "T.I." Harris) returns home to...	0	0	0	0	1	0	0	0	0	0	0	0	
30080	A grieving widower moves to the country where ...	A grieving widower moves to the country where ...	0	0	0	0	0	0	0	1	0	0	0	0	
30081	A passionate make-up artist	A passionate make-up	0	0	0	0	0	1	0	0	0	0	0	0	

In this dataset we have 2 type of plots, Long and Short. I wanted to create a setup where I will use plot_short to train one model and plot_long to train the other model but I feel it's better to train the model with plot_long as it has more characters.

So I will be removing the plot_short from the dataset so that it will be ready for Pre Processing

In [ ]:
```
df_2 = df_2.drop(['plot_short'], axis=1)
df_2
```

Out[ ]:

	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horror	music	musi
0	Francis, a	0	0	0	0	0	0	0	0	0	1	0	0	1	0	

	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horror	music	musi
	young man, recalls in his memory th...															
1	In 16th-century Prague, a rabbi creates the Go...	0	0	0	0	0	0	0	0	0	1	0	0	1	0	
2	The callous rich, portrayed by Lennox, think o...	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
3	Based on the Robert Louis Stevenson story: Doc...	0	0	0	0	0	0	0	1	0	0	0	0	1	0	
4	Southern negro Sylvia Landry visits her cousin...	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
30077	The story is about friendship and family	1	0	0	0	1	0	0	0	0	0	0	0	0	0	

	plot_long	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horror	music	musi
	witch...															
30078	Blue Story is a tragic tale of a friendship be...	0	0	0	0	0	1	0	1	0	0	0	0	0	0	
30079	A good man (Tip "T.I." Harris) returns home to...	0	0	0	0	1	0	0	0	0	0	0	0	0	0	
30080	A grieving widower moves to the country where ...	0	0	0	0	0	0	0	1	0	0	0	0	0	0	
	A passionate make-up															

Our encoded dataset is ready to be exported!

In [ ]:
```
df_2.to_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/df_2.csv')
```

Movie Genre Prediction(MGP) using NLP

Hemansh Anand

Notebook 2: Pre Processing

The main purpose of this notebook is to Pre Process the data and make it ready for Modeling. We begin by creating a function(plot_cleaner) that cleans the plot. After this we split the Dataset into train and test and then Vectorize our independant variable to feed it to the model.

This notebook accomplishes three primary tasks:

1. Cleaning the Plot(our independant variable)
2. Applying TF-IDF Vectorizer
3. Splitting the data into train and test sets for modeling.

Import the Libraries and Dataset

In [ ]:

```
import pandas as pd
import numpy as np
import json
import nltk
import re
import csv
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split

%matplotlib inline
# pd.set_option('display.max_colwidth', 300)
```

In [ ]:

```
df = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/df_2.csv', index_col=0)
df.rename(columns={'plot_long':'plot'}, inplace=True)
df.head()
```

Out[ ]:

	plot	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horror	music	musical
0	Francis, a young man, recalls in his memory th...	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0
1	In 16th-century	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0

	plot	action	adventure	animation	biography	comedy	crime	documentary	drama	family	fantasy	film-noir	history	horror	music	musical
	Prague, a rabbi creates the Go...															
2	The callous rich, portrayed by Lennox, think o...	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
3	Based on the Robert Louis Stevenson story: Doc...	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0
4	Southern negro Sylvia Landry	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0

As I have encoded the genres before NLP Pre Processing, hence it is a good practice to make the genre columns look different from the features that will be created.

Hence I will be adding gen_ before all the genres so that later I don't get confused.

In [ ]:

```
genres = list(df.columns.values)
genre_val = genres[1:]
genre_val = ['gen_'+g for g in genre_val]
print(len(genre_val))
print(genre_val)
```

```
22
```

```
['gen_action', 'gen_adventure', 'gen_animation', 'gen_biography', 'gen_comedy', 'gen_crime', 'gen_documentary', 'gen_
drama', 'gen_family', 'gen_fantasy', 'gen_film-noir', 'gen_history', 'gen_horror', 'gen_music', 'gen_musical', 'gen_m
```

In [ ]:

```
new_col_names = ['plot'] + genre_val

print(new_col_names)
```

```
['plot', 'gen_action', 'gen_adventure', 'gen_animation', 'gen_biography', 'gen_comedy', 'gen_crime', 'gen_documentary
', 'gen_drama', 'gen_family', 'gen_fantasy', 'gen_film-noir', 'gen_history', 'gen_horror', 'gen_music', 'gen_musical
', 'gen_mystery', 'gen_romance', 'gen_sci-fi', 'gen_sport', 'gen_thriller', 'gen_war', 'gen_western']
```

Let's apply the new column names

In [ ]:

```
df.columns = new_col_names

df.head()
```

Out[ ]:

	plot	gen_action	gen_adventure	gen_animation	gen_biography	gen_comedy	gen_crime	gen_documentary	gen_drama	gen_family	gen_fanta
0	Francis, a young man, recalls in his memory th...	0	0	0	0	0	0	0	0	0	
1	In 16th-century Prague, a rabbi creates the Go...	0	0	0	0	0	0	0	0	0	
2	The callous rich, portrayed by Lennox, think o...	0	0	0	0	0	0	0	1	0	

	plot	gen_action	gen_adventure	gen_animation	gen_biography	gen_comedy	gen_crime	gen_documentary	gen_drama	gen_family	gen_fanta
3	Based on the Robert Louis Stevenson story: Doc... Southern negro Sylvia	0	0	0	0	0	0	0	1	0	

It's time to seperate the Independant Variables and Target Variables from the dataset.

- X contains plots
- y contains genres

In [ ]:
```
X = df.drop(genre_val, axis=1)
y = df[genre_val]
```

In [ ]:
```
X.head(1)
```

Out[ ]:

	plot
0	Francis, a young man, recalls in his memory th...

In [ ]:
```
y.head(1)
```

Out[ ]:

	gen_action	gen_adventure	gen_animation	gen_biography	gen_comedy	gen_crime	gen_documentary	gen_drama	gen_family	gen_fantasy	gen_filn no
0	0	0	0	0	0	0	0	0	0	1	

Text Cleaning

In [ ]:

```
import nltk
nltk.download('stopwords')
nltk.download('punkt')
nltk.download('wordnet')
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
from nltk.stem import WordNetLemmatizer
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Helper Functions

In [ ]:

```
# Tokenize
def my_tokenizer(text):
    return word_tokenize(text)

# Stop Words
stop_words = stopwords.words('english')
stop_words.extend(['.',','])
def my_remove_stopwords(list_of_tokens):

    cleaned_tokens = []

    for token in list_of_tokens:
        if token in stop_words: continue
        cleaned_tokens.append(token)

    return cleaned_tokens

#Stem
def my_stemmer(list_of_tokens):

    stemmed_tokens_list = []

    for i in list_of_tokens:

        token = PorterStemmer().stem(i)
        stemmed_tokens_list.append(token)

    return stemmed_tokens_list
```

```
    lemmatized_tokens_list = []

    for i in list_of_tokens:
        token = WordNetLemmatizer().lemmatize(i)
        lemmatized_tokens_list.append(token)

    return lemmatized_tokens_list

#Untokenize
def my_untokenizer(token_list):
        return " ".join(token_list)
```

Plot Cleaner

In [ ]:
```
def plot_cleaner(text):

    plot_cleaned = []
    num_texts = len(text)

    for i in text.index:
        that_string = text[i]
        tokenized_list = my_tokenizer(that_string)
        removed_stopwords = my_remove_stopwords(tokenized_list)
        stemmed_words = my_stemmer(removed_stopwords)
        lemmatized_words = my_lemmatizer(stemmed_words)
        untokenize = my_untokenizer(lemmatized_words)

        plot_cleaned.append(untokenize)

    return plot_cleaned
```

Let's apply our text_cleaner to the plots

In [ ]:
```
X['cleaned_plot'] = plot_cleaner(X['plot'])
```

Let's compare the changes done by the text_cleaner

In [ ]:
```
X['plot'][0]
```

Out[ ]:
```
"Francis, a young man, recalls in his memory the horrible experiences he and his fiancée Jane recently went through.
It is the annual fair in Holstenwall. Francis and his friend Alan visit The Cabinet of Dr. Caligari, an exhibit where
the mysterious doctor shows the somnambulist Cesare, and awakens him for some moments from his death-like sleep. When
Alan asks Cesare about his future, Cesare answers that he will die before dawn. The next morning Alan is found dead.
Francis suspects Cesare of being the murderer, and starts spying on him and Dr. Caligari. The following night Cesare
is going to stab Jane in her bed, but softens when he sees the beautiful woman, and instead of committing another mur
der, he abducts her. Jane's father awakens because of the noise, and he and some servants follow the fleeing Cesare.
```

```
When Cesare cannot outrun his pursuers anymore, he gently places Jane down on the ground, and runs away. Francis and the police investigate the caravan of Dr. Caligari, but the ..."
```

In [ ]:
```
X['cleaned_plot'][0]
```

Out[ ]:
```
"franci young man recal memori horribl experi fiancé jane recent went It annual fair holstenwal franci friend alan visit the cabinet dr. caligari exhibit mysteri doctor show somnambulist cesar awaken moment death-lik sleep when alan ask cesar futur cesar answer die dawn the next morn alan found dead franci suspect cesar murder start spi dr. caligari the follow night cesar go stab jane bed soften see beauti woman instead commit anoth murder abduct jane 's father awaken nois servant follow flee cesar when cesar outrun pursuer anymor gentli place jane ground run away franci polic investig caravan dr. caligari ..."
```

The changes look good, we can drop the orignal plot from our dataset now

In [ ]:
```
X = X.drop(['plot'],axis=1)

X.head()
```

Out[ ]:

	cleaned_plot
0	franci young man recal memori horribl experi f...
1	In 16th-centuri pragu rabbi creat golem - gian...
2	the callou rich portray lennox think pleasur a...
3	base robert loui stevenson stori : doctor henr...
4	southern negro sylvia landri visit cousin alma...

Before vectorizing, we will split the data into train and test

We are ready with clean plot, next step is to split the dataset into train and test before vectorizing the clean text.

As we have enough data, I decided to simpy use a 75;25 train-test split.

Experiment Setup 1

For the Exp_1, I have decided to train the model with two different TF-IDF Vectorizers by changing the parameters as I wanted to experiment with it.

1. Variation 1(min_df at 0.005 and max_df at 0.95)
2. Variation 2(min_df at 20)

Variation 1

(min_df at 0.005 and max_df at 0.95)

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = 123)
```

In [ ]:
```
# Variation 1
from sklearn.feature_extraction.text import TfidfVectorizer
# min_df at 0.005 and max_df at 0.95
tfidf = TfidfVectorizer(max_df=0.95, min_df=0.005)



tfidf.fit(X_train['cleaned_plot']) # Fit the vectorizer to the training data

# Transform both train and test sets
X_train_tfidf = tfidf.transform(X_train['cleaned_plot'])
X_test_tfidf = tfidf.transform(X_test['cleaned_plot'])

import joblib
joblib.dump(tfidf, 'tfidf_min_max.pkl')
```

Out[ ]:
```
['tfidf_min_max.pkl']
```

In [ ]:
```
X_train_tfidf
```

Out[ ]:
```
<22533x1621 sparse matrix of type '<class 'numpy.float64'>'
```

```
        with 682342 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
X_test_tfidf
```

Out[ ]:
```
<7512x1621 sparse matrix of type '<class 'numpy.float64'>'
        with 225334 stored elements in Compressed Sparse Row format>
```

Let's convert this sparse matrix back to dataframe format

In [ ]:
```
X_train_vectorized_df = pd.DataFrame(X_train_tfidf.toarray(), index=X_train.index,
columns=tfidf.get_feature_names())
X_train_vectorized_df
```

Out[ ]:

	000	10	12	15	16	1970	20	30	abandon	abduct	abil	abl	abus	accept	accid	accident	accompani	account	accus	achiev	across
27154	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.151140
11719	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
10596	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
6445	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
12540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.145262
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28673	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
17752	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
28066	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
15740	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
19994	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

22533 rows × 1621 columns

In [ ]:
```
X_test_vectorized_df = pd.DataFrame(X_test_tfidf.toarray(), index=X_test.index,
columns=tfidf.get_feature_names())
X_test_vectorized_df
```

Out[ ]:

	000	10	12	15	16	1970	20	30	abandon	abduct	abil	abl	abus	accept	accid	accident	accompani	account	accus	achiev	acr
16894	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
21356	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
13514	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
8981	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
16232	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.158626	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14126	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
3776	0.154948	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
8040	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
13152	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	
8535	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	

7512 rows × 1621 columns

In [ ]:
```
X_train_vectorized_df_full = X_train.join(X_train_vectorized_df, on=X_train_vectorized_df.index)
X_test_vectorized_df_full = X_test.join(X_test_vectorized_df, on=X_test_vectorized_df.index)
```

It's time to merge the genre encoded values with these vectors so that we can export this later

In [ ]:

```
train_df_rte = X_train_vectorized_df_full.drop(['cleaned_plot'], axis=1).copy()
train_df_rte = train_df_rte.merge(y_train, on=train_df_rte.index)
train_df_rte = train_df_rte.set_index(['key_0'])
train_df_rte.reset_index(drop=True, inplace=True)
train_df_rte
```

Out[ ]:

	000	10	12	15	16	1970	20	30	abandon	abduct	abil	abl	abus	accept	accid	accident	accompani	account	accus	achiev	across
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.151140
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	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.145262
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22528	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
22529	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
22530	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
22531	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
22532	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

22533 rows × 1643 columns

In [ ]:

```
test_df_rte = X_test_vectorized_df_full.drop(['cleaned_plot'], axis=1).copy()
test_df_rte = test_df_rte.merge(y_test, on=test_df_rte.index)
test_df_rte = test_df_rte.set_index(['key_0'])
test_df_rte.reset_index(drop=True, inplace=True)
test_df_rte
```

Out[ ]:

	000	10	12	15	16	1970	20	30	abandon	abduct	abil	abl	abus	accept	accid	accident	accompani	account	accus	achiev	acro
0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
1	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
2	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
3	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
4	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.158626	0.0	0.0	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7507	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
7508	0.154948	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
7509	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
7510	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0
7511	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	0

7512 rows × 1643 columns

Export the Train and Test Datasets

In [ ]:
```
train_df_rte.to_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup 1/train_min_max.csv')
test_df_rte.to_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup 1/test_min_max.csv')
```

Variation 2

(min_df at 20)

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25, random_state = 123)
```

In [ ]:
```
# Variation 2
from sklearn.feature_extraction.text import TfidfVectorizer
# min_df at 20
tfidf = TfidfVectorizer(min_df=20)



tfidf.fit(X_train['cleaned_plot']) # Fit the vectorizer to the training data

# Transform both train and test sets
X_train_tfidf = tfidf.transform(X_train['cleaned_plot'])
X_test_tfidf = tfidf.transform(X_test['cleaned_plot'])

import joblib
joblib.dump(tfidf, 'tfidf_min_20.pkl')
```

Out[ ]:
```
['tfidf_min_20.pkl']
```

In [ ]:
```
X_train_tfidf
```

Out[ ]:
```
<22533x5493 sparse matrix of type '<class 'numpy.float64'>'
        with 859922 stored elements in Compressed Sparse Row format>
```

In [ ]:
```
X_test_tfidf
```

Out[ ]:
```
<7512x5493 sparse matrix of type '<class 'numpy.float64'>'
        with 283716 stored elements in Compressed Sparse Row format>
```

Let's convert this sparse matrix back to dataframe format

In [ ]:
```
X_train_vectorized_df = pd.DataFrame(X_train_tfidf.toarray(), index=X_train.index,
columns=tfidf.get_feature_names())
X_train_vectorized_df
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	16th	17	17th	18	18th	19	1900	1920	1930	1939	1940	1941	1942	1943	1944
27154	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
11719	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
10596	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
6445	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
12540	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
28673	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
17752	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
28066	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
15740	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
19994	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

22533 rows × 5493 columns

In [ ]:
```
X_test_vectorized_df = pd.DataFrame(X_test_tfidf.toarray(), index=X_test.index,
columns=tfidf.get_feature_names())
X_test_vectorized_df
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	16th	17	17th	18	18th	19	1900	1920	1930	1939	1940	1941	1942	1943	1
16894	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
21356	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13514	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8981	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
16232	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
14126	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3776	0.0	0.122397	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8040	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
13152	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
8535	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

7512 rows × 5493 columns

In [ ]:
```
X_train_vectorized_df_full = X_train.join(X_train_vectorized_df, on=X_train_vectorized_df.index)
X_test_vectorized_df_full = X_test.join(X_test_vectorized_df, on=X_test_vectorized_df.index)
```

It's time to merge the genre encoded values with these vectors so that we can export this later

In [ ]:

```
train_df_rte = X_train_vectorized_df_full.drop(['cleaned_plot'], axis=1).copy()
train_df_rte = train_df_rte.merge(y_train, on=train_df_rte.index)
train_df_rte = train_df_rte.set_index(['key_0'])
train_df_rte.reset_index(drop=True, inplace=True)
train_df_rte
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	16th	17	17th	18	18th	19	1900	1920	1930	1939	1940	1941	1942	1943	1944
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
22528	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22529	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22530	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22531	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
22532	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

22533 rows × 5515 columns

In [ ]:

```
test_df_rte = X_test_vectorized_df_full.drop(['cleaned_plot'], axis=1).copy()
test_df_rte = test_df_rte.merge(y_test, on=test_df_rte.index)
test_df_rte = test_df_rte.set_index(['key_0'])
test_df_rte.reset_index(drop=True, inplace=True)
test_df_rte
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	16th	17	17th	18	18th	19	1900	1920	1930	1939	1940	1941	1942	1943	19
0	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	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	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
2	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
3	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
4	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
7507	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7508	0.0	0.122397	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7509	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7510	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	
7511	0.0	0.000000	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	

7512 rows × 5515 columns

Export the Train and Test Datasets

In [ ]:

```
train_df_rte.to_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup
1/train_min_20.csv')
test_df_rte.to_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup
1/test_min_20.csv')
```

Movie Genre Prediction(MGP) using NLP

Hemansh Anand

Notebook 3: Modeling with Exp1_Var1

The main purpose of this notebook is to fit the model using the train data and find out the best reults. We have first done scaling of the Data and then used OneVsRest with Logistic Regression,Stochastic Gradient Descent and MultinomialNB.

This notebook accomplishes two primary tasks:

1. Fit and Export different models.
2. Discuss the best results.

Import Libraraies

In [1]:

```
#import libraries
import pandas as pd
import numpy as np
import json
import nltk
import re
import csv
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import joblib

from ast import literal_eval

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

from sklearn.metrics import f1_score
```

Import the Dataset

Importing Train and Test dataframes from the previous notebook.

In [2]:
```
train = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup 1/train_min_max.csv', index_col=0)
test = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Datasets/Experiment Setup 1/test_min_max.csv', index_col=0)
```

In [3]:
```
train.head()
```

Out[3]:

	000	10	12	15	16	1970	20	30	abandon	abduct	abil	abl	abus	accept	accid	accident	accompani	account	accus	achiev	across	act
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.151140	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	0.0	0.0	0.000000	0.0
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000	0.0
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.145262	0.0

5 rows × 1643 columns

Splitting Data into X and y

- X contains the features

- y contains the genres

In [5]:
```
cols = list(train.columns.values)
genre_cols = cols[-22:]
print(len(genre_cols))
print(genre_cols)
```

```
22
['gen_action', 'gen_adventure', 'gen_animation', 'gen_biography', 'gen_comedy', 'gen_crime', 'gen_documentary', 'gen_
drama', 'gen_family', 'gen_fantasy', 'gen_film-noir', 'gen_history', 'gen_horror', 'gen_music', 'gen_musical', 'gen_m
ystery', 'gen_romance', 'gen_sci-fi', 'gen_sport', 'gen_thriller', 'gen_war', 'gen_western']
```

In [6]:
```
X_train = train[train.columns[~train.columns.isin(genre_cols)]]
y_train = train[train.columns[ train.columns.isin(genre_cols)]]


X_test = test[test.columns[~test.columns.isin(genre_cols)]]
y_test = test[test.columns[ test.columns.isin(genre_cols)]]
```

Experiment Setup 2 Scaling the Data

- Exp_2_Var_1 Standard Scaler
- Exp_2_Var_2 Min Max Scaler

Exp_2_Var_1 Standard Scaler

In [6]:
```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler().fit(X_train)
X_train_std = std_scaler.transform(X_train)
X_test_std = std_scaler.transform(X_test)

joblib.dump(std_scaler, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers
/standard_scaler.pkl')
```

Out[6]:
```
['/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers/standard_scaler.pkl']
```

Exp_3 Different Models under Standard Scaling

- Exp_3_Var_1 Logistic Regression
- Exp_3_Var_2 Stochastic Gradient Descent

Exp_3_Var_1 Logistic Regression

In [7]:

```
model_1 = OneVsRestClassifier(LogisticRegression(random_state=123, max_iter=3000, C=1, n_jobs=-1),
n_jobs=-1).fit(X_train_std, y_train)

# export the model
joblib.dump(model_1, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max
/logreg.pkl')

# Make predictions
y_train_model_1 = model_1.predict(X_train_std)
y_pred_model_1 = model_1.predict(X_test_std)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_1, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_1, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_1, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_1, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_1, average='weighted')
print('Train F1 Score:' train_f1_score)
```

```
train_classification_report= classification_report(y_test, y_pred_model_1)

print(train_classification_report)
```

```
Train Recall Score: 0.618053935131171
Test Recall Score: 0.5011467889908257
Train Precision Score: 0.7651395078018712
Test Precision Score: 0.5943196500485064
Train F1 Score: 0.6770202480236456
Test F1 Score: 0.5389817539871439
              precision    recall  f1-score   support

           0       0.59      0.44      0.51      1349
           1       0.56      0.36      0.44       867
           2       0.23      0.25      0.24       258
           3       0.37      0.27      0.31       376
           4       0.65      0.55      0.60      2650
           5       0.65      0.45      0.53      1240
           6       0.48      0.60      0.53       334
           7       0.71      0.75      0.73      4197
           8       0.39      0.29      0.33       483
           9       0.48      0.28      0.35       600
          10       0.25      0.20      0.22        81
          11       0.32      0.28      0.30       351
          12       0.67      0.54      0.60      1003
          13       0.54      0.38      0.45       465
          14       0.17      0.18      0.17       219
          15       0.46      0.28      0.35       685
          16       0.62      0.44      0.51      1656
          17       0.60      0.48      0.53       648
          18       0.40      0.49      0.44       199
          19       0.61      0.46      0.53      1833
          20       0.44      0.50      0.47       395
          21       0.42      0.49      0.45       167

   micro avg       0.60      0.50      0.54     20056
   macro avg       0.48      0.41      0.44     20056
weighted avg       0.59      0.50      0.54     20056
 samples avg       0.61      0.55      0.54     20056
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Exp_3_Var_2 Stochastic Gradient Descent

In [8]:

```
model_2 = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l2', random_state=123, alpha=1,
n_jobs=-1, max_iter=3000), n_jobs=-1).fit(X_train_std, y_train)

# export the model
joblib.dump(model_2, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/SGD.pkl')

# Make predictions
y_train_model_2 = model_2.predict(X_train_std)
y_pred_model_2 = model_2.predict(X_test_std)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_2, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_2, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_2, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_2, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_2, average='weighted')
print('Train F1 Score:',train_f1_score)
test_f1_score = f1_score(y_test, y_pred_model_2, average='weighted')
```

```
print(test_classification_report)
```

```
Train Recall Score: 0.22930764358811961
Test Recall Score: 0.21604507379337853
Train Precision Score: 0.764186259886078
Test Precision Score: 0.713901988503582
Train F1 Score: 0.24904694944436337
Test F1 Score: 0.23355985891861222
              precision    recall  f1-score   support

           0       1.00      0.03      0.06      1349
           1       0.00      0.00      0.00       867
           2       0.00      0.00      0.00       258
           3       0.00      0.00      0.00       376
           4       0.84      0.21      0.34      2650
           5       0.84      0.05      0.10      1240
           6       1.00      0.03      0.06       334
           7       0.69      0.82      0.75      4197
           8       0.00      0.00      0.00       483
           9       1.00      0.00      0.00       600
          10       0.00      0.00      0.00        81
          11       0.00      0.00      0.00       351
          12       0.97      0.07      0.13      1003
          13       1.00      0.01      0.02       465
          14       0.00      0.00      0.00       219
          15       0.50      0.00      0.00       685
          16       0.94      0.01      0.02      1656
          17       0.97      0.05      0.09       648
          18       0.00      0.00      0.00       199
          19       0.83      0.05      0.09      1833
          20       0.92      0.03      0.06       395
          21       0.00      0.00      0.00       167

   micro avg       0.72      0.22      0.33     20056
   macro avg       0.52      0.06      0.08     20056
weighted avg       0.71      0.22      0.23     20056
 samples avg       0.53      0.27      0.34     20056

CPU times: user 3.2 s, sys: 2.53 s, total: 5.73 s
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Exp_2_Var_2 Min-Max Scaler

In [7]:

```
from sklearn.preprocessing import MinMaxScaler

mima_scaler = MinMaxScaler().fit(X_train)

X_train_mima = mima_scaler.transform(X_train)

X_test_mima = mima_scaler.transform(X_test)


joblib.dump(mima_scaler, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers
/mima_scaler.pkl')
```

Out[7]:

```
['/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers/mima_scaler.pkl']
```

Exp_4 Different Models under Min-Max Scaling

- Exp_4_Var_1 Logistic Regression
- Exp_4_Var_2 Stochastic Gradient Descent
- Exp_4_Var_3 Multinomial Naive Bayes

Exp_4_Var_1 Logistic Regression

In [8]:

```
model_4 = OneVsRestClassifier(LogisticRegression(random_state=123, max_iter=3000, C=1, n_jobs=-1),
n_jobs=-1).fit(X_train_mima, y_train)

# export the model
joblib.dump(model_4, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max
/logreg_mima.pkl')

# Make predictions
y_train_model_4 = model_4.predict(X_train_mima)
y_pred_model_4 = model_4.predict(X_test_mima)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_4, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_4, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_4, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_4, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_4, average='weighted')
print('Train F1 Score:' train_f1_score)
```

```
train_classification_report= classification_report(y_test, y_pred_model_4)

print(train_classification_report)
```

```
Train Recall Score: 0.5261342044734825
Test Recall Score: 0.4624052652572796
Train Precision Score: 0.7780862290973529
Test Precision Score: 0.6777482104217563
Train F1 Score: 0.6053594198166531
Test F1 Score: 0.530830645439008
              precision    recall  f1-score   support

           0       0.64      0.41      0.50      1349
           1       0.65      0.30      0.41       867
           2       0.50      0.07      0.12       258
           3       0.61      0.15      0.24       376
           4       0.67      0.54      0.59      2650
           5       0.70      0.41      0.52      1240
           6       0.87      0.48      0.62       334
           7       0.72      0.76      0.74      4197
           8       0.66      0.16      0.26       483
           9       0.63      0.20      0.30       600
          10       0.00      0.00      0.00        81
          11       0.58      0.15      0.24       351
          12       0.76      0.49      0.60      1003
          13       0.77      0.31      0.44       465
          14       0.68      0.09      0.15       219
          15       0.53      0.23      0.32       685
          16       0.66      0.42      0.51      1656
          17       0.78      0.41      0.54       648
          18       0.77      0.36      0.49       199
          19       0.63      0.44      0.52      1833
          20       0.74      0.44      0.55       395
          21       0.87      0.29      0.43       167

   micro avg       0.69      0.46      0.55     20056
   macro avg       0.66      0.32      0.41     20056
weighted avg       0.68      0.46      0.53     20056
 samples avg       0.67      0.52      0.54     20056
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Exp_4_Var_2 Stochastic Gradient Descent

In [9]:

```
model_6 = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l2', random_state=123, alpha=1,
n_jobs=-1, max_iter=3000), n_jobs=-1).fit(X_train_mima, y_train)

# EXPORT THE MODEL
joblib.dump(model_6, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max
/sgd_mima.pkl')

# Make predictions
y_train_model_6 = model_6.predict(X_train_mima)
y_pred_model_6 = model_6.predict(X_test_mima)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_6, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_6, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_6, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_6, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_6, average='weighted')
print('Train F1 Score:',train_f1_score)
```

```
test_classification_report= classification_report(y_test, y_pred_model_6)

print(test_classification_report)
```

```
Train Recall Score: 0.20922364078802627
Test Recall Score: 0.20926406063023534
Train Precision Score: 0.11655724327928345
Test Precision Score: 0.11691710096713227
Train F1 Score: 0.14971124453127138
Test F1 Score: 0.15001815056197756
              precision    recall  f1-score   support

           0       0.00      0.00      0.00      1349
           1       0.00      0.00      0.00       867
           2       0.00      0.00      0.00       258
           3       0.00      0.00      0.00       376
           4       0.00      0.00      0.00      2650
           5       0.00      0.00      0.00      1240
           6       0.00      0.00      0.00       334
           7       0.56      1.00      0.72      4197
           8       0.00      0.00      0.00       483
           9       0.00      0.00      0.00       600
          10       0.00      0.00      0.00        81
          11       0.00      0.00      0.00       351
          12       0.00      0.00      0.00      1003
          13       0.00      0.00      0.00       465
          14       0.00      0.00      0.00       219
          15       0.00      0.00      0.00       685
          16       0.00      0.00      0.00      1656
          17       0.00      0.00      0.00       648
          18       0.00      0.00      0.00       199
          19       0.00      0.00      0.00      1833
          20       0.00      0.00      0.00       395
          21       0.00      0.00      0.00       167

   micro avg       0.56      0.21      0.30     20056
   macro avg       0.03      0.05      0.03     20056
weighted avg       0.12      0.21      0.15     20056
 samples avg       0.56      0.25      0.33     20056
```

```
CPU times: user 3.13 s, sys: 2.57 s, total: 5.7 s
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Exp_4_Var_3 Multinomial Naive Bayes

In [11]:

```
model_7 = OneVsRestClassifier(MultinomialNB(alpha=1), n_jobs=-1).fit(X_train_mima, y_train)

# export the model
joblib.dump(model_7, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max
/MNB_mima.pkl')

# Make predictions
y_train_model_7 = model_7.predict(X_train_mima)
y_pred_model_7 = model_7.predict(X_test_mima)

# Check overall accuracies
# from sklearn.metrics import accuracy_score
# train_acc = accuracy_score(y_train, y_train_model_1)
# test_acc = accuracy_score(y_test, y_pred_model_1)
# train_scores.append(train_acc)
# test_scores.append(test_acc)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_7, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_7, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_7, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_7, average='weighted')
```

```
train_f1_score = f1_score(y_train, y_train_model_7, average='weighted')
print('Train F1 Score:',train_f1_score)

test_f1_score = f1_score(y_test, y_pred_model_7, average='weighted')
print('Test F1 Score:',test_f1_score)

# Classification Report
train_classification_report= classification_report(y_test, y_pred_model_7)
print(train_classification_report)
```

```
Train Recall Score: 0.4832327744258142
Test Recall Score: 0.4426106900678101
Train Precision Score: 0.7074829158767606
Test Precision Score: 0.6672724992731439
Train F1 Score: 0.5479590699648026
Test F1 Score: 0.5053991466956003
              precision    recall  f1-score   support

           0       0.60      0.44      0.51      1349
           1       0.59      0.27      0.37       867
           2       0.48      0.05      0.09       258
           3       0.49      0.15      0.23       376
           4       0.66      0.51      0.58      2650
           5       0.68      0.39      0.50      1240
           6       0.73      0.45      0.55       334
           7       0.71      0.78      0.74      4197
           8       0.72      0.05      0.10       483
           9       0.59      0.15      0.24       600
          10       0.00      0.00      0.00        81
          11       0.52      0.19      0.28       351
          12       0.78      0.41      0.54      1003
          13       0.80      0.22      0.34       465
          14       0.86      0.03      0.05       219
          15       0.52      0.17      0.26       685
```

```
           16       0.65      0.38      0.48      1656
           17       0.72      0.35      0.47       648
           18       0.88      0.26      0.40       199
           19       0.65      0.43      0.52      1833
           20       0.71      0.45      0.55       395
           21       0.92      0.14      0.25       167

   micro avg       0.68      0.44      0.53     20056
   macro avg       0.65      0.29      0.37     20056
weighted avg       0.67      0.44      0.51     20056
 samples avg       0.64      0.50      0.52     20056

CPU times: user 9.65 s, sys: 3.29 s, total: 12.9 s
Wall time: 10.3 s
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Results

Standard Scaler

In [12]:

```
results_std = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Results/TFIDF min max/Standard
Scaler_min_max.csv', index_col=0)
results_std
```

Out[12]:

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.594320	0.501147	0.538982	0.765140	0.618054	0.677020
SGD	0.713902	0.216045	0.233560	0.764186	0.229308	0.249047

Min Max Scaler

In [3]:
```
results_mima = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Results/TFIDF min max/Min Max Scaler_min_max.csv', index_col=0)
results_mima
```

Out[3]:

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.677748	0.462405	0.530831	0.778086	0.526134	0.605359
Stochastic Gradient Descent	0.116917	0.209264	0.150018	0.116557	0.209224	0.149711
MultinomialNB	0.667272	0.442611	0.505399	0.707483	0.483233	0.547959

Conclusion

After comparing all the models above with their respective scores, it was found that Min Max scaling is better than Standard Scaling for our dataset.

Among the models trained after Min Max scaling, Logistic Regression performed the best with the highest F1 score.

We will now perform modeling on the Exp1_Var2 and find out the best results and then compare Exp1_Var1 and Exp1_Var2.

Movie Genre Prediction(MGP) using NLP

Hemansh Anand

Notebook 3: Modeling with Exp1_Var2

The main purpose of this notebook is to fit the model using the train data and find out the best reults. We have first done scaling of the Data and then used OneVsRest with Logistic Regression,Stochastic Gradient Descent and MultinomialNB.

This notebook accomplishes two primary tasks:

1. Fit and Export different models.
2. Discuss the best results.

Import Libraraies

In [2]:

```
#import libraries
import pandas as pd
import numpy as np
import json
import nltk
import re
import csv
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import joblib

from ast import literal_eval

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
```

Import the Dataset

Importing Train and Test dataframes from the previous notebook.

In [ ]:
```
train = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Modified Datasets/train.csv',
index_col=0)
test = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Modified Datasets/test.csv',
index_col=0)
```

In [ ]:
```
train.head()
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	17	17th	18	18th	19	1900	1920	1930	1936	1939	1940	1941	1942	1943	1944
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.269463
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

5 rows × 5473 columns

Splitting Data into X and y

- X contains the features
- y contains the genres

In [ ]:
```
cols = list(train.columns.values)
genre_cols = cols[-22:]
print(len(genre_cols))
print(genre_cols)
```

```
22
['gen_action', 'gen_adventure', 'gen_animation', 'gen_biography', 'gen_comedy', 'gen_crime', 'gen_documentary', 'gen_
drama', 'gen_family', 'gen_fantasy', 'gen_film-noir', 'gen_history', 'gen_horror', 'gen_music', 'gen_musical', 'gen_m
ystery', 'gen_romance', 'gen_sci-fi', 'gen_sport', 'gen_thriller', 'gen_war', 'gen_western']
```

In [ ]:
```
X_train = train[train.columns[~train.columns.isin(genre_cols)]]
y_train = train[train.columns[ train.columns.isin(genre_cols)]]

X_test = test[test.columns[~test.columns.isin(genre_cols)]]
y_test = test[test.columns[ test.columns.isin(genre_cols)]]
```

Experiment Setup 2 Scaling the Data

- Exp_2_Var_1 Standard Scaler
- Exp_2_Var_2 Min Max Scaler

Exp_2_Var_1 Standard Scaler

In [ ]:
```
from sklearn.preprocessing import StandardScaler
std_scaler = StandardScaler().fit(X_train)
X_train_std = std_scaler.transform(X_train)
X_test_std = std_scaler.transform(X_test)

joblib.dump(std_scaler, 'standard_scaler.pkl')
```

Out[ ]: ['standard_scaler.pkl']

Exp_3 Different Models under Standard Scaling

- Exp_3_Var_1 Logistic Regression
- Exp_3_Var_2 Stochastic Gradient Descent

Exp_3_Var_1 Logistic Regression

In [ ]:

```
model_1 = OneVsRestClassifier(LogisticRegression(random_state=123, max_iter=3000, C=1, n_jobs=-1),
n_jobs=-1).fit(X_train_std, y_train)

# export the mode;
joblib.dump(model_1, '/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/model_1.pkl')

# Make predictions
y_train_model_1 = model_1.predict(X_train_std)
y_pred_model_1 = model_1.predict(X_test_std)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_1, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_1, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_1, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_1, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_1, average='weighted')
test_f1_score = f1_score(y_test, y_pred_model_1, average='weighted')
```

Exp_3_Var_2 Stochastic Gradient Descent

In [ ]:

```
model_2 = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l2', random_state=123, alpha=1,
n_jobs=-1, max_iter=3000), n_jobs=-1).fit(X_train_std, y_train)

# export the model
joblib.dump(model_2, '/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/model_2.pkl')

# Make predictions
y_train_model_2 = model_2.predict(X_train_std)
y_pred_model_2 = model_2.predict(X_test_std)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_2, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_2, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_2, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_2, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_2, average='weighted')
test_f1_score = f1_score(y_test, y_pred_model_2, average='weighted')
```

Exp_2_Var_2 Min-Max Scaler

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
mima_scaler = MinMaxScaler().fit(X_train)
X_train_mima = mima_scaler.transform(X_train)
X_test_mima = mima_scaler.transform(X_test)


joblib.dump(mima_scaler, 'mima_scaler.pkl')
```

Out[ ]:
```
['mima_scaler.pkl']
```

Exp_4 Different Models under Min-Max Scaling

- Exp_4_Var_1 Logistic Regression
- Exp_4_Var_2 Stochastic Gradient Descent
- Exp_4_Var_3 Multinomial Naive Bayes

Exp_4_Var_1 Logistic Regression

In [ ]:

```
model_4 = OneVsRestClassifier(LogisticRegression(random_state=123, max_iter=3000, C=1, n_jobs=-1),
n_jobs=-1).fit(X_train_mima, y_train)

# export the model
joblib.dump(model_4, '/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/model_4.pkl')

# Make predictions
y_train_model_4 = model_4.predict(X_train_mima)
y_pred_model_4 = model_4.predict(X_test_mima)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_4, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_4, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_4, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_4, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_4, average='weighted')
test_f1_score = f1_score(y_test, y_pred_model_4, average='weighted')
```

```
Train Recall Score: 0.6398425905854497
Test Recall Score: 0.49018728830444314
Train Precision Score: 0.8694203140770593
Test Precision Score: 0.6891000561352323
              precision    recall  f1-score   support

           0       0.69      0.48      0.57      1404
           1       0.64      0.32      0.43       862
           2       0.51      0.13      0.21       232
           3       0.62      0.19      0.29       392
           4       0.69      0.58      0.63      2702
           5       0.70      0.44      0.54      1208
           6       0.88      0.50      0.64       357
           7       0.71      0.76      0.73      4137
           8       0.61      0.18      0.27       446
           9       0.61      0.26      0.37       564
          10       0.00      0.00      0.00        81
          11       0.61      0.21      0.32       330
          12       0.80      0.55      0.65      1019
          13       0.85      0.31      0.46       494
          14       0.57      0.09      0.15       227
          15       0.53      0.23      0.32       686
          16       0.65      0.43      0.52      1699
          17       0.82      0.45      0.58       617
          18       0.85      0.34      0.49       219
          19       0.63      0.47      0.54      1847
          20       0.83      0.43      0.57       384
          21       0.90      0.31      0.46       169

   micro avg       0.70      0.49      0.58     20076
   macro avg       0.67      0.35      0.44     20076
weighted avg       0.69      0.49      0.56     20076
 samples avg       0.68      0.54      0.56     20076

/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Exp_4_Var_2 Stochastic Gradient Descent

In [ ]:

```
model_6 = OneVsRestClassifier(SGDClassifier(loss='hinge', penalty='l2', random_state=123, alpha=1,
n_jobs=-1, max_iter=3000), n_jobs=-1).fit(X_train_mima, y_train)

# export the model
joblib.dump(model_6, '/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/model_6.pkl')

# Make predictions
y_train_model_6 = model_6.predict(X_train_mima)
y_pred_model_6 = model_6.predict(X_test_mima)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_6, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_6, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_6, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_6, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_6, average='weighted')
test_f1_score = f1_score(y_test, y_pred_model_6, average='weighted')
```

```
Train Recall Score: 0.21028497106935018
Test Recall Score: 0.20606694560669456
Train Precision Score: 0.11770022503020618
Test Precision Score: 0.11350006044133876
```

	precision	recall	f1-score	support
0	0.00	0.00	0.00	1404
1	0.00	0.00	0.00	862
2	0.00	0.00	0.00	232
3	0.00	0.00	0.00	392
4	0.00	0.00	0.00	2702
5	0.00	0.00	0.00	1208
6	0.00	0.00	0.00	357
7	0.55	1.00	0.71	4137
8	0.00	0.00	0.00	446
9	0.00	0.00	0.00	564
10	0.00	0.00	0.00	81
11	0.00	0.00	0.00	330
12	0.00	0.00	0.00	1019
13	0.00	0.00	0.00	494
14	0.00	0.00	0.00	227
15	0.00	0.00	0.00	686
16	0.00	0.00	0.00	1699
17	0.00	0.00	0.00	617
18	0.00	0.00	0.00	219
19	0.00	0.00	0.00	1847
20	0.00	0.00	0.00	384
21	0.00	0.00	0.00	169
micro avg	0.55	0.21	0.30	20076
macro avg	0.03	0.05	0.03	20076
weighted avg	0.11	0.21	0.15	20076
samples avg	0.55	0.24	0.32	20076

```
CPU times: user 6.56 s, sys: 6.1 s, total: 12.7 s
Wall time: 31.4 s
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
```

```
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to co
ntrol this behavior.
```

Exp_4_Var_3 Multinomial Naive Bayes

In [ ]:

```
model_7 = OneVsRestClassifier(MultinomialNB(alpha=1), n_jobs=-1).fit(X_train_mima, y_train)

# export the model
joblib.dump(model_7, f'/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/model_7.pkl')

# Make predictions
y_train_model_7 = model_7.predict(X_train_mima)
y_pred_model_7 = model_7.predict(X_test_mima)

from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

# Recall Score
train_recall_score = recall_score(y_train, y_train_model_7, average='weighted')
test_recall_score = recall_score(y_test, y_pred_model_7, average='weighted')
print('Train Recall Score:',train_recall_score)
print('Test Recall Score:',test_recall_score)

# Precision Score
train_precision_score = precision_score(y_train, y_train_model_7, average='weighted')
test_precision_score = precision_score(y_test, y_pred_model_7, average='weighted')
print('Train Precision Score:',train_precision_score)
print('Test Precision Score:',test_precision_score)

# F1 Score
train_f1_score = f1_score(y_train, y_train_model_7, average='weighted')
print('Train F1 Score:',train_f1_score)
```

```
train_classification_report= classification_report(y_test, y_pred_model_7)

print(train_classification_report)
```

```
Train Recall Score: 0.622284103983592
Test Recall Score: 0.5328750747160789
Train Precision Score: 0.7240758473671053
Test Precision Score: 0.6524731443418311
Train F1 Score: 0.6590790049701707
Test F1 Score: 0.5738608522714271
              precision    recall  f1-score   support

           0       0.63      0.56      0.59      1404
           1       0.55      0.41      0.47       862
           2       0.33      0.12      0.18       232
           3       0.47      0.31      0.37       392
           4       0.69      0.61      0.65      2702
           5       0.66      0.50      0.57      1208
           6       0.68      0.61      0.64       357
           7       0.72      0.78      0.75      4137
           8       0.64      0.17      0.26       446
           9       0.55      0.30      0.39       564
          10       0.00      0.00      0.00        81
          11       0.44      0.34      0.39       330
          12       0.76      0.59      0.66      1019
          13       0.79      0.29      0.42       494
          14       0.56      0.04      0.07       227
          15       0.54      0.25      0.34       686
          16       0.61      0.46      0.53      1699
          17       0.67      0.54      0.60       617
          18       0.82      0.28      0.42       219
          19       0.63      0.53      0.57      1847
          20       0.68      0.56      0.61       384
          21       0.91      0.37      0.52       169

   micro avg       0.66      0.53      0.59     20076
   macro avg       0.61      0.39      0.45     20076
weighted avg       0.65      0.53      0.57     20076
 samples avg       0.67      0.59      0.58     20076
```

```
CPU times: user 28.6 s, sys: 9.08 s, total: 37.7 s
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Results

Standard Scaler

In [ ]:
```
results_std = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/Results/Standard Scaler.csv', index_col=0)
results_std
```

Out[ ]:

	Test Precission Score	Test Recall Score	Train F1 Score	Train Precission Score	Train Recall Score	Test F1 Score
Logistic Regression	0.513606	0.527496	0.520458	0.916105	0.899735	0.907846
SGD	0.741086	0.274856	0.400991	0.800719	0.306948	0.443778

Min Max Scaler

In [ ]:
```
results_mima = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained
Models/Results/MinMax Scaler.csv', index_col=0)
results_mima
```

Out[ ]:

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.689100	0.490187	0.572868	0.869420	0.639843	0.737171

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
SGD	0.113500	0.206067	0.146377	0.117700	0.210285	0.150925

Conclusion

After comparing all the models above with their respective scores, it was found that Min Max scaling is better than Standard Scaling for our dataset.

Among the models trained after Min Max scaling, Stochastic Gradient Descent performed the worst. Multinomial Naive Bayes and Logistic Regression performed the equally good and I would like to try some hyperparameter tuning on both of them to find the better model.

Movie Genre Prediction(MGP) using NLP

Hemansh Anand

Notebook 4: Hyperparameter Optimization and Cross Validation

The main purpose of this notebook is to compare the results of previous experiments. After comparison, the top 2 models have been optimized using hyperparameters. Later the best model is cross validated and the best results are then discussed.

This notebook accomplishes three primary tasks:

1. Fit and Export the best model.
2. Discuss the best results.
3. Evaluate the overall attempt and outcome.

Import Libraraies

In [ ]:

```
#import libraries
import pandas as pd
import numpy as np
import json
import nltk
import re
import csv
import matplotlib.pyplot as plt
import seaborn as sns
from tqdm import tqdm
import joblib

from ast import literal_eval

from sklearn.multiclass import OneVsRestClassifier
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.linear_model import LogisticRegression
from sklearn.linear_model import SGDClassifier

from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV

from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.pipeline import Pipeline

from sklearn.metrics import f1_score
```

```
from sklearn.metrics import multilabel_confusion_matrix
from sklearn.metrics import confusion_matrix
```

We will now compare the results of our first Experiment_Setup_1. We had experimented with two parameters for TF-IDF Vectorization which are given below:

- Variation 1 (min_df=0.005, max_df=)
- Variation 2 (min_df=20)

Best models from both the Variations have been shortlisted and are presented here for comparison.

Best model from Variation 1

In [ ]:
```
results_exp1_var1 = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Results/TFIDF min max/Best_Result_min_max.csv', index_col=0)
results_exp1_var1
```

Out[ ]:

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.677748	0.462405	0.530831	0.778086	0.526134	0.605359

Best model from Experiment 1 Variation 2

In [ ]:
```
results_exp1_var2 = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Results/TFIDF min_20/Best Results min_20.csv', index_col=0)
results_exp1_var2
```

Out[ ]:

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score

	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.689100	0.490187	0.572868	0.869420	0.639843	0.737171

After comparing the results from both the variations, it is clear that Experiment 1 Variation 2 has performed better. It is now time to perform some hyperparameter tuning to these two models to find out which one is superior and shall be used in our Movie Genre Prediction.

Import the Dataset

Importing Train and Test dataframes from the previous notebook.

In [ ]:
```
train = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Modified Datasets/train.csv',
index_col=0)
test = pd.read_csv('/content/drive/MyDrive/Colab Files/Movie Genre/Modified Datasets/test.csv',
index_col=0)
```

In [ ]:
```
train.head()
```

Out[ ]:

	00	000	10	100	11	12	13	13th	14	15	16	17	17th	18	18th	19	1900	1920	1930	1936	1939	1940	1941	1942	1943	1944
0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
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	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.269463
3	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000
4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.000000

5 rows × 5473 columns

Splitting Data into X and y

- X contains the features

- y contains the genres

In [ ]:
```
cols = list(train.columns.values)
genre_cols = cols[-22:]
print(len(genre_cols))
print(genre_cols)
```

```
22
['gen_action', 'gen_adventure', 'gen_animation', 'gen_biography', 'gen_comedy', 'gen_crime', 'gen_documentary', 'gen_drama', 'gen_family', 'gen_fantasy', 'gen_film-noir', 'gen_history', 'gen_horror', 'gen_music', 'gen_musical', 'gen_mystery', 'gen_romance', 'gen_sci-fi', 'gen_sport', 'gen_thriller', 'gen_war', 'gen_western']
```

In [ ]:
```
X_train = train[train.columns[~train.columns.isin(genre_cols)]]
y_train = train[train.columns[ train.columns.isin(genre_cols)]]

X_test = test[test.columns[~test.columns.isin(genre_cols)]]
y_test = test[test.columns[ test.columns.isin(genre_cols)]]
```

In [ ]:
```
from sklearn.preprocessing import MinMaxScaler
mima_scaler = MinMaxScaler().fit(X_train)
X_train_mima = mima_scaler.transform(X_train)
X_test_mima = mima_scaler.transform(X_test)

joblib.dump(mima_scaler, '/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers/mima_scaler.pkl')
```

Out[ ]:
```
['/content/drive/MyDrive/Colab Files/MGP/Trained Models/TFIDF_min_max/Scalers/mima_scaler.pkl']
```

Expermient 3 Hyperparameter Optimization

- Logistic Regression with c = [0.01, 0.1, 1]
- Multinomial Naive Bayes with alpha = [0.01, 0.1, 1]

Logistic Regression with c = [0.01, 0.1, 1]

In [ ]:

```
c_val = [0.01, 0.1, 1]

for c in c_val:
    model_1 = OneVsRestClassifier(LogisticRegression(random_state=123, max_iter=3000, C=c,
n_jobs=-1), n_jobs=-1).fit(X_train_mima, y_train)

    # export the model
    joblib.dump(model_1, f'/content/drive/MyDrive/Colab Files/MGP/Trained Models/parameter
tuning/legreg_mima_{c}.pkl')

    # Make predictions
    y_train_model_1 = model_1.predict(X_train_mima)
    y_pred_model_1 = model_1.predict(X_test_mima)

    from sklearn.metrics import classification_report, recall_score, precision_score, f1_score

    # Recall Score
    print(f'C:  {c}')
    train_recall_score = recall_score(y_train, y_train_model_1, average='weighted')
    test_recall_score = recall_score(y_test, y_pred_model_1, average='weighted')
    print('Train Recall Score:',train_recall_score)
    print('Test Recall Score:',test_recall_score)

    # Precision Score
    train_precision_score = precision_score(y_train, y_train_model_1, average='weighted')
    test_precision_score = precision_score(y_test, y_pred_model_1, average='weighted')
    print('Train Precision Score:',train_precision_score)
    print('Test Precision Score:',test_precision_score)
```

```
    test_f1_score = f1_score(y_test, y_pred_model_1, average='weighted')
    print('Test F1 Score:',test_f1_score)
```

```
C:  0.01
Train Recall Score: 0.20634973570559104
Test Recall Score: 0.2001892807332138
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Train Precision Score: 0.5351997345963526
Test Precision Score: 0.4354281120894809
Train F1 Score: 0.18630677161001544
Test F1 Score: 0.17880294856425905
C:  0.1
Train Recall Score: 0.4059795567857798
Test Recall Score: 0.3671050009962144
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
Train Precision Score: 0.8301835546790085
Test Precision Score: 0.7587101351235627
Train F1 Score: 0.4844188022924335
Test F1 Score: 0.4393929814834858
C:  1
Train Recall Score: 0.6398425905854497
Test Recall Score: 0.49018728830444314
Train Precision Score: 0.8694203140770593
Test Precision Score: 0.6891000561352323
Train F1 Score: 0.7211126264442678
Test F1 Score: 0.556423947353071
```

Multinomial Naive Bayes with alpha = [0.01, 0.1, 1]

In [ ]:

```
a_val = [0.01, 0.1, 1]

for a in a_val:
    model_7 = OneVsRestClassifier(MultinomialNB(alpha=a), n_jobs=-1).fit(X_train_mima, y_train)

    # export the model
    joblib.dump(model_7, f'/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained Models/MNB
/model_7_{a}.pkl')

    # Make predictions
    y_train_model_7 = model_7.predict(X_train_mima)
    y_pred_model_7 = model_7.predict(X_test_mima)

    from sklearn.metrics import classification_report, recall_score, precision_score

    train_recall_score = recall_score(y_train, y_train_model_7, average='weighted')
    test_recall_score = recall_score(y_test, y_pred_model_7, average='weighted')
    print(f'Alpha:  {a}')
    print('Train Recall Score:',train_recall_score)
    print('Test Recall Score:',test_recall_score)

    train_precision_score = precision_score(y_train, y_train_model_7, average='weighted')
    test_precision_score = precision_score(y_test, y_pred_model_7, average='weighted')
    print('Train Precision Score:',train_precision_score)
    print('Test Precision Score:',test_precision_score)

    # F1 Score
    train_f1_score = f1_score(y_train, y_train_model_7, average='weighted')
```

```
Alpha:  0.01
Train Recall Score: 0.6729252472028147
Test Recall Score: 0.5346184498904164
Train Precision Score: 0.7304812057913629
Test Precision Score: 0.6457024044004941
Train F1 Score: 0.6977081837684609
Test F1 Score: 0.5751450907183039
Alpha:  0.1
Train Recall Score: 0.6694068799919961
Test Recall Score: 0.5445805937437737
Train Precision Score: 0.7249121807851051
Test Precision Score: 0.6464274162821652
Train F1 Score: 0.6929876705225838
Test F1 Score: 0.5825155984904712
Alpha:  1
Train Recall Score: 0.622284103983592
Test Recall Score: 0.5328750747160789
Train Precision Score: 0.7240758473671053
Test Precision Score: 0.6524731443418311
Train F1 Score: 0.6590790049701707
Test F1 Score: 0.5738608522714271
CPU times: user 1min 31s, sys: 27.7 s, total: 1min 59s
Wall time: 2min 28s
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Results

In [ ]:

```
results_tuning = pd.read_csv('/content/drive/MyDrive/Colab Files/MGP/Results/After parameter tuning.csv', index_col=0)
results_tuning
```

Out[ ]:

Classifier	Alpha/C	Test Precission Score	Test Recall Score	Test F1 Score	Train Precission Score	Train Recall Score	Train F1 Score
Logistic Regression	0.01	0.435428	0.200189	0.178803	0.535200	0.206350	0.186307
Logistic Regression	0.10	0.758710	0.367105	0.439393	0.830184	0.405980	0.484419
Logistic Regression	1.00	0.689100	0.490187	0.556424	0.869420	0.639843	0.721113
MultnimoialNB	0.01	0.645702	0.534618	0.575145	0.730481	0.672925	0.697708
MultnimoialNB	0.10	0.646427	0.544581	0.582516	0.724912	0.669407	0.692988
MultnimoialNB	1.00	0.652473	0.532875	0.573861	0.724076	0.622284	0.659079

From the above results, it acn be seen that MultnimoialNB with alpha=0.10 performs the best.

Cross Validation

Multinomial Naive Bayes (alpha=0.10)

In [ ]:

```
from sklearn.model_selection import cross_val_score
model_12 = OneVsRestClassifier(MultinomialNB(alpha=0.1), n_jobs=-1)

scores = cross_val_score(model_12, X_train_mima, y_train, cv = 5)

for i in range(len(scores)) :
    print(f"Fold {i+1}: {scores[i]}")
print(f"The average score is :{np.mean(scores)}")
```

```
Fold 1: 0.1335700022187708
Fold 2: 0.12250332889480692
Fold 3: 0.13382157123834887
Fold 4: 0.13559698180204172
Fold 5: 0.12960497114957834
The average score is :0.13101937106070932
```

In [ ]:
```
model_12 = OneVsRestClassifier(MultinomialNB(alpha=0.1), n_jobs=-1).fit(X_train_mima, y_train)


# export and save model
joblib.dump(model_12, '/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained Models/Cross Validated Models/MNB_0.1.pkl')
```

Out[ ]:
```
['/content/drive/MyDrive/Colab Files/Movie Genre/Final Trained Models/Cross Validated Models/MNB_0.1.pkl']
```

Discussion of Best Results

The measure of best results for this project is the highest F1 score on the test set.

After experimenting with 4 Different Setups:

1. TF-IDF (two variations)
2. Scaling (two variations)
3. 3 Different Classification Models
4. Hyper Parameter Tuning between the top 2 models

I conclude that the best model is OneVsRest with Multnimoial Naive Bayes using TF-IDF plot vectors (min_df=20) and Min Max Scaling.

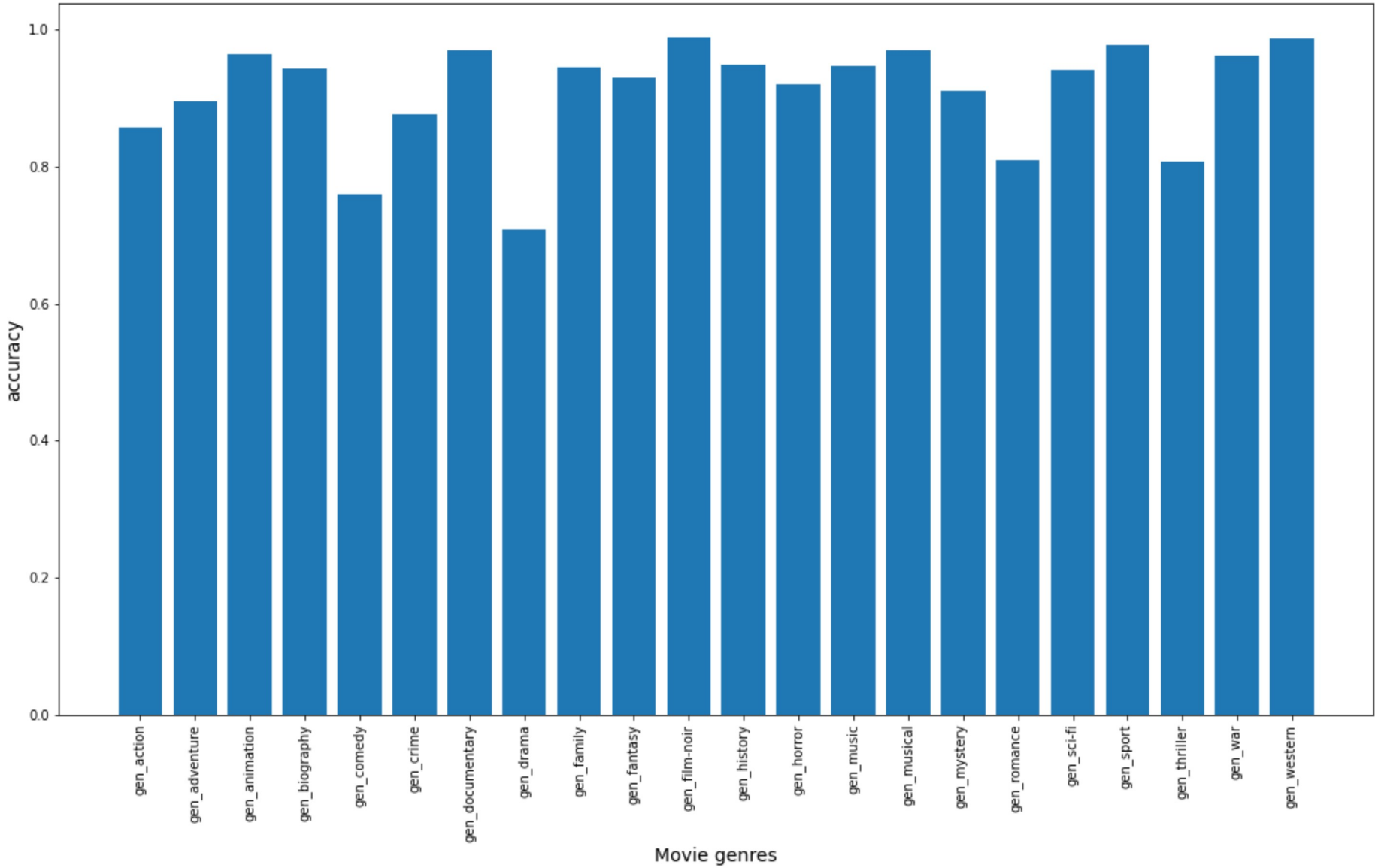
Yes, I had to retrain the model with different parameters during the experimentation.

Below are the main conclusions:

- MultnimoialNB showed good accuracy rate. The lowest accuracy rate is around 70% and the highest accuracy rate is around 98%.
- MultnimoialNB is much faster to implement than other models and it is my preferred choice for deploying in the Movie Genre Prediction.

In [ ]:

```
plt.figure(figsize=(18,10))
p1 =plt.bar(scores_df.Genres, height=scores_df.Scores)
plt.xticks( rotation=90)
# plt.title("Movies genre classification accuracy (multinomialNB VS LinearSVC)")
plt.ylabel('accuracy', fontsize=14)
plt.xlabel('Movie genres', fontsize=14)
# plt.legend(p1[0]), ('Scores'))
plt.show()
```


1.0
0.8
0.6
0.4
0.2
0.0
accuracy
gen_action
gen_adventure
gen_animation
gen_biography
gen_comedy
gen_crime
gen_documentary
gen_drama
gen_family
gen_fantasy
gen_film-noir
gen_history
gen_horror
gen_music
gen_musical
gen_mystery
gen_romance
gen_sci-fi
gen_sport
gen_thriller
gen_war
gen_western
Movie genres

In [ ]:
```
y_train_pred = model_12.predict(X_train_mima)
y_test_pred = model_12.predict(X_test_mima)
y_train_proba = model_12.predict_proba(X_train_mima)
y_test_proba = model_12.predict_proba(X_test_mima)
```

In [ ]:
```
from sklearn.metrics import accuracy_score, f1_score
print(f'Training F1 score: {f1_score(y_train, y_train_pred,average="weighted"):0.4f}')
print(f'    Test F1 score: {f1_score(y_test, y_test_pred,average="weighted"):0.4f}')
print(f'Training Accuracy Score: {accuracy_score(y_train, y_train_pred):0.4f}')
print(f'    Test Accuracy Score: {accuracy_score(y_test, y_test_pred):0.4f}')
```

```
Training F1 score: 0.6930
    Test F1 score: 0.5825
Training Accuracy Score: 0.2189
    Test Accuracy Score: 0.1373
```

Confusion Matrix

In [ ]:
```
cmatrix = multilabel_confusion_matrix(y_test, y_test_pred)


gen_cm_list = []
for gen in cmatrix:
    gen_cm_list.append(pd.DataFrame(gen, columns=[' Positive', ' Negative  '], index=
['Positive','Negative']))


gen_cm_list[10].values
```

Out[ ]:
```
array([[7426,    4],
       [  77,    4]])
```

In [ ]:
```
cf_matrix=gen_cm_list[10].values
```

In [ ]:

```
y_pred_df = pd.DataFrame(y_test_pred, columns=genre_cols)
score=[]
# Test set predictions
for gen in genre_cols:
    score.append([accuracy_score(y_test[gen], y_pred_df[gen])])
print(score)
```

```
[[0.8576754093995473], [0.8945546531753429], [0.9640527226734124], [0.9428837704699774], [0.7595526561043803], [0.876
3147383837039], [0.968845692983624], [0.7085607775262948], [0.9440820130475303], [0.9298362401810678], [0.98921581680
20238], [0.9487418452935694], [0.919185195047264], [0.9471441885234989], [0.9696445213686593], [0.9101318066835308],
[0.8092131540407402], [0.940354147250699], [0.9781653574757023], [0.8081480495273599], [0.9628544800958594], [0.98681
93316469178]]
```

In [ ]:

```
scores_df = pd.DataFrame.from_records(score)
gen_df = pd.DataFrame({'Genre':genre_cols})
```

In [ ]:

```
scores_df.insert(1,"Genre",genre_cols,True)
```

In [ ]:

```
scores_df.columns = ['Scores', 'Genres']
scores_df
```

Out[ ]:

	Scores	Genres
0	0.857675	gen_action
1	0.894555	gen_adventure
2	0.964053	gen_animation
3	0.942884	gen_biography
4	0.759553	gen_comedy
5	0.876315	gen_crime
6	0.968846	gen_documentary

	Scores	Genres
7	0.708561	gen_drama
8	0.944082	gen_family
9	0.929836	gen_fantasy
10	0.989216	gen_film-noir
11	0.948742	gen_history
12	0.919185	gen_horror
13	0.947144	gen_music
14	0.969645	gen_musical
15	0.910132	gen_mystery
16	0.809213	gen_romance
17	0.940354	gen_sci-fi
18	0.978165	gen_sport
19	0.808148	gen_thriller
20	0.962854	gen_war

We now take a closer look at the complete classification report for each genre in this model.

In [ ]:

```
from sklearn.metrics import classification_report
test_classification_report= classification_report(y_test, y_test_pred, target_names=genre_cols)
train_classification_report= classification_report(y_train, y_train_pred, target_names=genre_cols)
print(train_classification_report)
print(test_classification_report)
```

```
                precision    recall  f1-score   support

    gen_action       0.69      0.67      0.68      4086
 gen_adventure       0.63      0.59      0.61      2477
 gen_animation       0.68      0.60      0.64       733
 gen_biography       0.63      0.57      0.60      1207
```

gen_comedy	0.72	0.68	0.70	7861
gen_crime	0.71	0.61	0.65	3710
gen_documentary	0.75	0.81	0.78	976
gen_drama	0.77	0.81	0.79	12611
gen_family	0.80	0.49	0.60	1363
gen_fantasy	0.69	0.50	0.58	1790
gen_film-noir	0.89	0.78	0.83	230
gen_history	0.61	0.66	0.63	1052
gen_horror	0.79	0.72	0.75	3013
gen_music	0.81	0.52	0.63	1375
gen_musical	0.84	0.51	0.63	647
gen_mystery	0.66	0.47	0.55	2090
gen_romance	0.66	0.59	0.62	5032
gen_sci-fi	0.76	0.71	0.73	1860
gen_sport	0.88	0.82	0.84	575
gen_thriller	0.70	0.62	0.66	5667
gen_war	0.73	0.76	0.75	1109
gen_western	0.91	0.82	0.87	507
micro avg	0.73	0.67	0.70	59971
macro avg	0.74	0.65	0.69	59971
weighted avg	0.72	0.67	0.69	59971
samples avg	0.74	0.71	0.68	59971

	precision	recall	f1-score	support
gen_action	0.63	0.57	0.60	1404
gen_adventure	0.55	0.43	0.48	862
gen_animation	0.37	0.24	0.29	232
gen_biography	0.44	0.32	0.37	392
gen_comedy	0.68	0.62	0.65	2702
gen_crime	0.65	0.50	0.56	1208
gen_documentary	0.70	0.61	0.65	357
gen_drama	0.72	0.77	0.75	4137
gen_family	0.58	0.21	0.31	446
gen_fantasy	0.55	0.34	0.42	564
gen_film-noir	0.50	0.05	0.09	81
gen_history	0.41	0.38	0.40	330
gen_horror	0.76	0.59	0.67	1019
gen_music	0.73	0.31	0.44	494
gen_musical	0.49	0.09	0.15	227
gen_mystery	0.51	0.28	0.36	686
gen_romance	0.60	0.48	0.53	1699
gen_sci-fi	0.66	0.55	0.60	617
gen_sport	0.75	0.37	0.50	219
gen_thriller	0.63	0.54	0.58	1847
gen_war	0.66	0.56	0.61	384

```
    gen_western       0.86      0.49      0.63       169

      micro avg       0.65      0.54      0.59     20076
      macro avg       0.61      0.42      0.48     20076
   weighted avg       0.65      0.54      0.58     20076
    samples avg       0.67      0.60      0.58     20076
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/metrics/_classification.py:1272: UndefinedMetricWarning: Precision and
F-score are ill-defined and being set to 0.0 in samples with no predicted labels. Use `zero_division` parameter to co
ntrol this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

Evaluation of Overall Attempt and Outcome

Yes, the orignal problem has been solved. The goals mentioned in the problem statement(Notebook 1) have been achieved.

Finally our model is now ready with a good F1 Score of 0.58

It's time to export the model and move to the next stage of deployment.

Export and Saving the model

In [ ]:
```
final_model = OneVsRestClassifier(MultinomialNB(alpha=0.10), n_jobs=-1).fit(X_train_mima, y_train)


# export and save the model for later use
joblib.dump(final_model, '/content/drive/MyDrive/Colab Files/MGP/Final
Model/final_model_MNB_0.10.pkl')
```

Out[ ]:
```
['/content/drive/MyDrive/Colab Files/MGP/Final Model/final_model_MNB_0.10.pkl']
```