Predicting Movie Genres using NLP - An Awesome Introduction to Multi-Label Classification

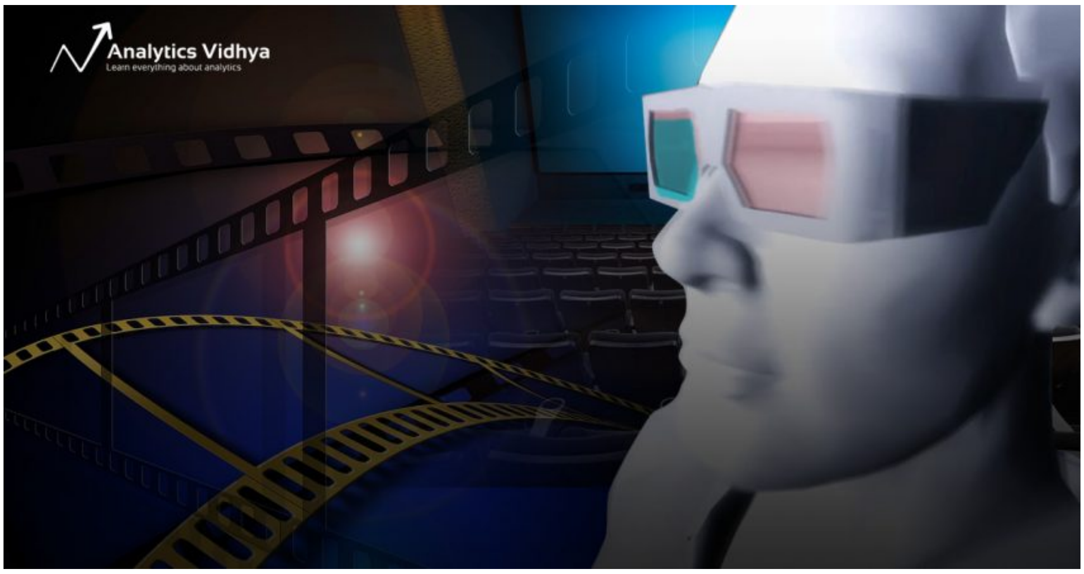
CLASSIFICATION MACHINE LEARNING NLP PROJECT PYTHON SUPERVISED TECHNIQUE TELECOM TEXT UNSTRUCTURED DATA

Introduction

I was intrigued going through this amazing article on building a multi-label image classification model last week. The data scientist in me started exploring possibilities of transforming this idea into a Natural Language Processing (NLP) problem.

That article showcases computer vision techniques to predict a movie's genre. So I had to find a way to convert that problem statement into text-based data. Now, most NLP tutorials look at solving single-label classification challenges (when there's only one label per observation).

But movies are not one-dimensional. One movie can span several genres. Now THAT is a challenge I love to embrace as a data scientist. I extracted a bunch of movie plot summaries and got down to work using this concept of multi-label classification. And the results, even using a simple model, are truly impressive.



In this article, we will take a very hands-on approach to understanding multi-label classification in NLP. I had a lot fun building the movie genre prediction model using NLP and I'm sure you will as well. Let's dig in!

Table of Contents

Brief Introduction to Multi-Label Classification

I'm as excited as you are to jump into the code and start building our genre classification model. Before we do that, however, let me introduce you to the concept of multi-label classification in NLP. *It's important to first understand the technique before diving into the implementation.*

The underlying concept is apparent in the name – multi-label classification. Here, an instance/record can have multiple labels and the number of labels per instance is not fixed.

Let me explain this using a simple example. Take a look at the below tables, where 'X' represents the input variables and 'y' represents the target variables (which we are predicting):

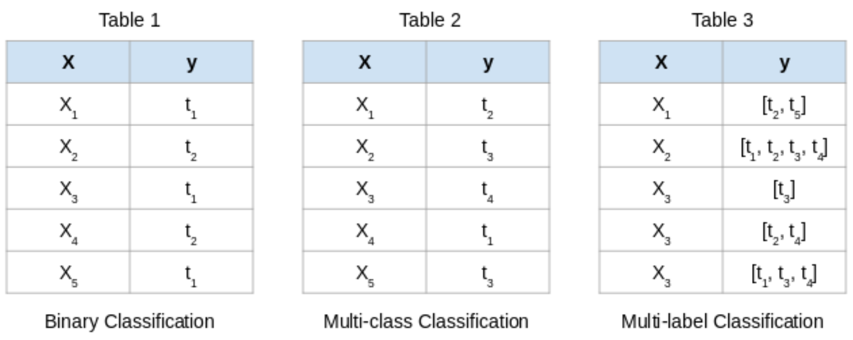


Table 1

X	y
X_1	t_1
X_2	t_2
X_3	t_1
X_4	t_2
X_5	t_1

Binary Classification

Table 2

X	y
X_1	t_2
X_2	t_3
X_3	t_4
X_4	t_1
X_5	t_3

Multi-class Classification

Table 3

X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_2, t_3, t_4]$
X_3	$[t_3]$
X_3	$[t_2, t_4]$
X_3	$[t_1, t_3, t_4]$

Multi-label Classification

- 'y' is a binary target variable in Table 1. Hence, there are only two labels – t1 and t2
- 'y' contains more than two labels in Table 2. But, notice how **there is *only one label for every input* in both these tables**
- You must have guessed why Table 3 stands out. We have multiple tags here, not just across the table, but for individual inputs as well

We cannot apply traditional classification algorithms directly on this kind of dataset. Why? Because these algorithms expect a single label for every input, when instead we have multiple labels. It's an intriguing challenge and one that we will solve in this article.

You can get a more in-depth understanding of multi-label classification problems in the below article:

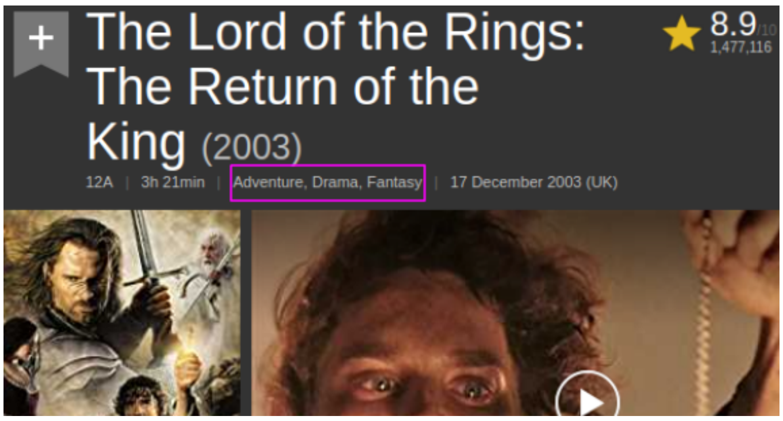
- Solving Multi-Label Classification Problems (using Case Studies)

Setting up our Multi-Label Classification Problem Statement

There are several ways of building a recommendation engine. When it comes to movie genres, you can slice and dice the data based on multiple variables. But here's a simple approach – build a model that can automatically predict genre tags! I can already imagine the possibilities of adding such an option to a recommender. A win-win for everyone.

Our task is to build a model that can predict the genre of a movie using just the plot details (available in text form).

Take a look at the below snapshot from IMDb and pick out the different things on display:


The Lord of the Rings: The Return of the King (2003)
8.9/10
1,477,116
12A | 3h 21min | Adventure, Drama, Fantasy | 17 December 2003 (UK)

There’s a LOT of information in such a tiny space:

- Movie title
- Movie rating in the top-right corner
- Total movie duration
- Release date
- And of course, the movie genres which I have highlighted in the magenta coloured bounding box

Genres tell us what to expect from the movie. And since these genres are clickable (at least on IMDb), they allow us to discover other similar movies of the same ilk. What seemed like a simple product feature suddenly has so many promising options.

About the Dataset

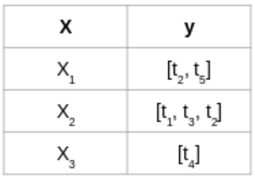
We will use the CMU Movie Summary Corpus open dataset for our project. **You can download the dataset directly from this link.**

This dataset contains multiple files, but we’ll focus on only two of them for now:

- **movie.metadata.tsv:** Metadata for 81,741 movies, extracted from the November 4, 2012 dump of Freebase. The movie genre tags are available in this file
- **plot_summaries.txt:** Plot summaries of 42,306 movies extracted from the November 2, 2012 dump of English-language Wikipedia. Each line contains the Wikipedia movie ID (which indexes into *movie.metadata.tsv*) followed by the plot summary

Our Strategy to Build a Movie Genre Prediction Model

We know that we can’t use supervised classification algorithms directly on a multi-label dataset. Therefore, we’ll first have to transform our target variable. Let’s see how to do this using a dummy dataset:

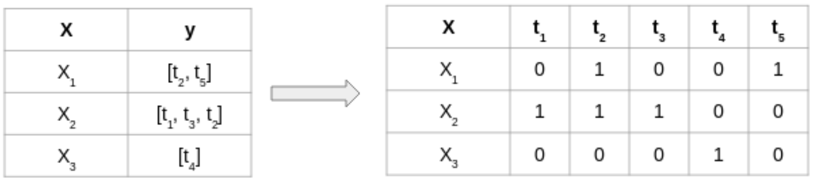


X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_3, t_2]$
X_3	$[t_4]$

Here, X and y are the features and labels, respectively – it is a multi-label dataset. Now, we will use the **Binary Relevance** approach to transform our target variable, y. We will first take out the unique labels in our dataset:

Unique labels = [t1, t2, t3, t4, t5]

There are 5 unique tags in the data. Next, we need to replace the current target variable with multiple target variables, each belonging to the unique labels of the dataset. Since there are 5 unique labels, there will be 5 new target variables with values 0 and 1 as shown below:



X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_3, t_2]$
X_3	$[t_4]$

X	t_1	t_2	t_3	t_4	t_5
X_1	0	1	0	0	1
X_2	1	1	1	0	0
X_3	0	0	0	1	0

We have now covered the necessary ground to finally start solving this problem. In the next section, we will finally make an **Automatic Movie Genre Prediction System** using Python!

Implementation: Using Multi-Label Classification to Build a Movie Genre Prediction Model (in Python)

We have understood the problem statement and built a logical strategy to design our model. Let's bring it all together and start coding!

Import the required libraries

We will start by importing the libraries necessary to our project:

```python
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)
```

view raw

genre_import_libraries.py hosted with ❤ by **GitHub**

Load Data

Let's load the movie metadata file first. Use '\t' as the separator as it is a tab separated file (.tsv):

```
meta = pd.read_csv("movie.metadata.tsv", sep = '\t', header = None) meta.head()
```

	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"}
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"}
4	261236	/m/01mrr1	A Woman in Flames	1983	NaN	106.0	{"/m/04306rv": "German Language"}	{"/m/0345h": "Germany"}	{"/m/07s9rl0": "Drama"}

Oh wait – there are no headers in this dataset. The first column is the **unique movie id**, the third column is the **name of the movie**, and the last column contains the **movie genre(s)**. We will not use the rest of the columns in this analysis.

Let's add column names to the aforementioned three variables:

```
# rename columns meta.columns = ["movie_id",1,"movie_name",3,4,5,6,7,"genre"]
```

Now, we will load the **movie plot dataset** into memory. This data comes in a text file with each row consisting of a movie id and a plot of the movie. We will read it line-by-line:

```
plots = []

with open("plot_summaries.txt", 'r') as f:
       reader = csv.reader(f, dialect='excel-tab')
       for row in tqdm(reader):
            plots.append(row)
```

view raw

genre_open_data.py hosted with ❤ by **GitHub**

Next, split the movie ids and the plots into two separate lists. We will use these lists to form a dataframe:

```
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})
```

view raw

genre_split_id_plot.py hosted with ❤ by **GitHub**

Let's see what we have in the 'movies' dataframe:

```
movies.head()
```

	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...

Perfect! We have both the movie id and the corresponding movie plot.

Data Exploration and Pre-processing

Let's add the movie names and their genres from the movie metadata file by merging the latter into the former based on the *movie_id* column:

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

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

movies.head()
```

view raw
genre_merge_data_labels.py hosted with ❤ by **GitHub**

	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"}

Great! We have added both movie names and genres. However, the genres are in a dictionary notation. It will be easier to work with them if we can convert them into a Python list. We'll do this using the first row:

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

Output:

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

We can't access the genres in this row by using just *.values( )*. Can you guess why? This is because this text is a string, not a dictionary. We will have to convert this string into a dictionary. We will take the help of the **json library** here:

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

Output:

```
dict
```

We can now easily access this row's genres:

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

Output:

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

This code helps us to extract all the genres from the movies data. Once done, add the extracted genres as lists back to the movies dataframe:

```
# 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
```

view raw
genre_extract_genres.py hosted with ❤ by GitHub

Some of the samples might not contain any genre tags. We should remove those samples as they won't play a part in our model building process:

```
# remove samples with 0 genre tags movies_new = movies[~(movies['genre_new'].str.len() == 0)]
```

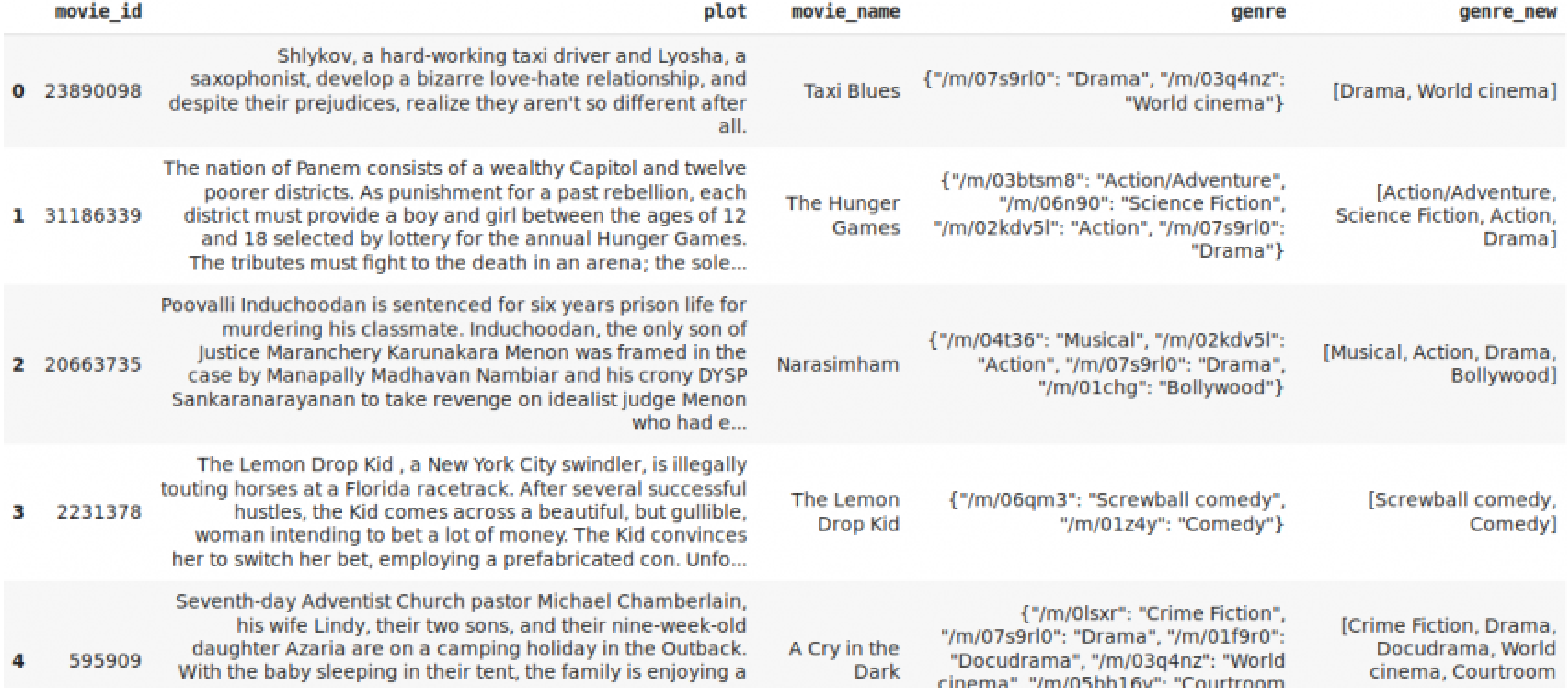
```
movies_new.shape, movies.shape
```

Output:

```
((41793, 5), (42204, 5))
```

Only 411 samples had no genre tags. Let's take a look at the dataframe once again:

```
movies.head()
```



	movie_id	plot	movie_name	genre	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	{"/m/07s9rl0": "Drama", "/m/03q4nz": "World cinema"}	[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	{"/m/03btsm8": "Action/Adventure", "/m/06n90": "Science Fiction", "/m/02kdv5l": "Action", "/m/07s9rl0": "Drama"}	[Action/Adventure, Science Fiction, Action, 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"}	[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	{"/m/06qm3": "Screwball comedy", "/m/01z4y": "Comedy"}	[Screwball comedy, 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	A Cry in the Dark	{"/m/0lsxr": "Crime Fiction", "/m/07s9rl0": "Drama", "/m/01f9r0": "Docudrama", "/m/03q4nz": "World cinema", "/m/05bh16v": "Courtroom	[Crime Fiction, Drama, Docudrama, World cinema, Courtroom

Notice that the genres are now in a list format. Are you curious to find how many movie genres have been covered in this dataset? The below code answers this question:

```
# get all genre tags in a list all_genres = sum(genres,[]) len(set(all_genres))
```

Output:

```
363
```

There are over 363 unique genre tags in our dataset. That is quite a big number. I can hardy recall 5-6 genres! Let's find out what are these tags. We will use **FreqDist( )** from the *nltk* library to create a dictionary of genres and their occurrence count across the dataset:

```
all_genres = nltk.FreqDist(all_genres)

# create dataframe
all_genres_df = pd.DataFrame({'Genre': list(all_genres.keys()),
                              'Count': list(all_genres.values())})
```

view raw

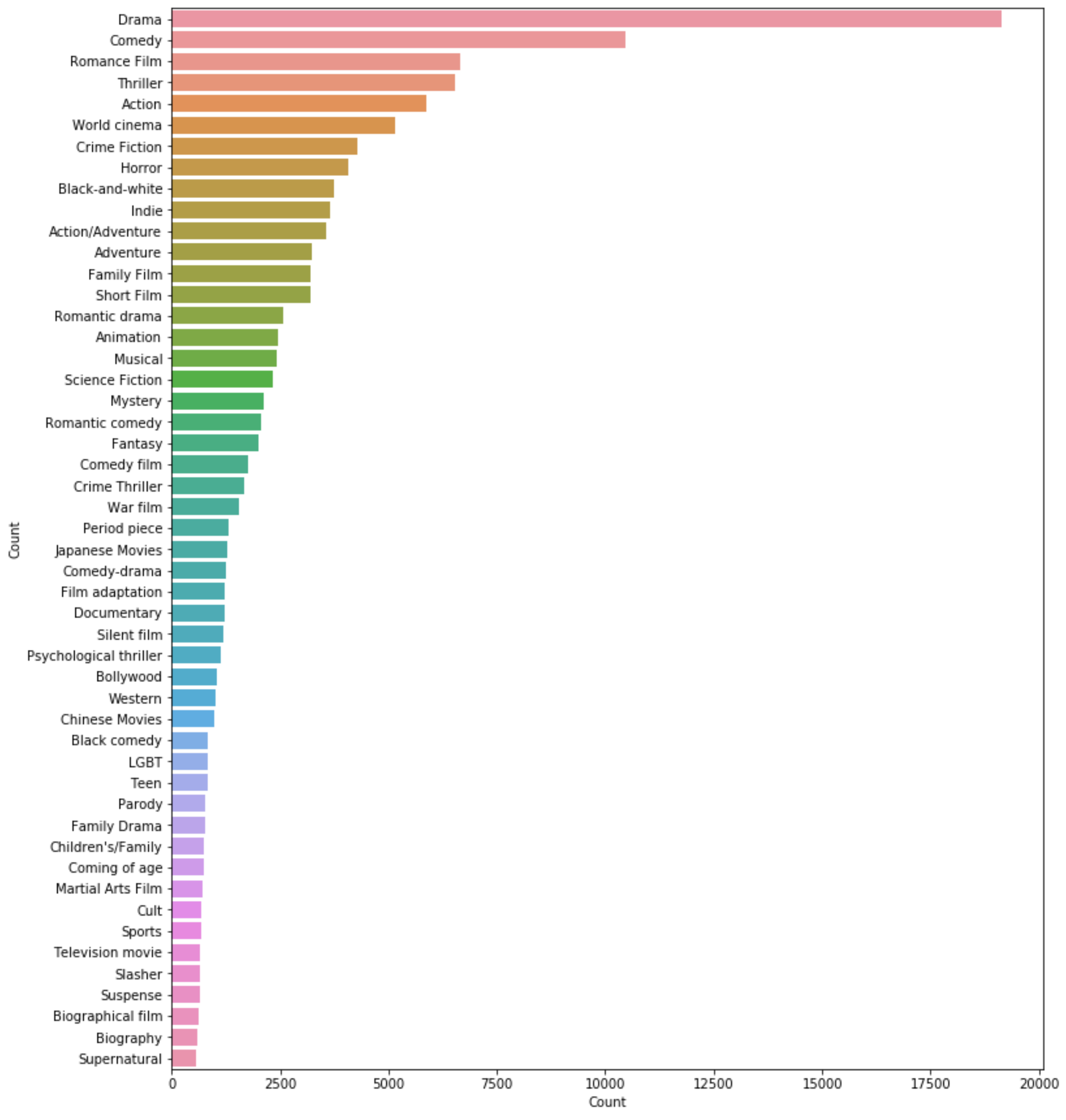
genre_frequencies.py hosted with ❤ by **GitHub**

I personally feel visualizing the data is a much better method than simply putting out numbers. So, let's plot the distribution of the movie genres:

```
g = all_genres_df.nlargest(columns="Count", n = 50)
plt.figure(figsize=(12,15))
ax = sns.barplot(data=g, x= "Count", y = "Genre")
ax.set(ylabel = 'Count')
plt.show()
```

view raw

genre_labels_visual.py hosted with ❤ by **GitHub**



Next, we will clean our data a bit. I will use some very basic text cleaning steps (as that is not the focus area of this article):

```
# function for text cleaning
def clean_text(text):
    # remove backslash-apostrophe
    text = re.sub("\'", "", text)
    # remove everything except alphabets
    text = re.sub("[^a-zA-Z]"," ",text)
    # remove whitespaces
    text = ' '.join(text.split())
    # convert text to lowercase
    text = text.lower()

    return text
```

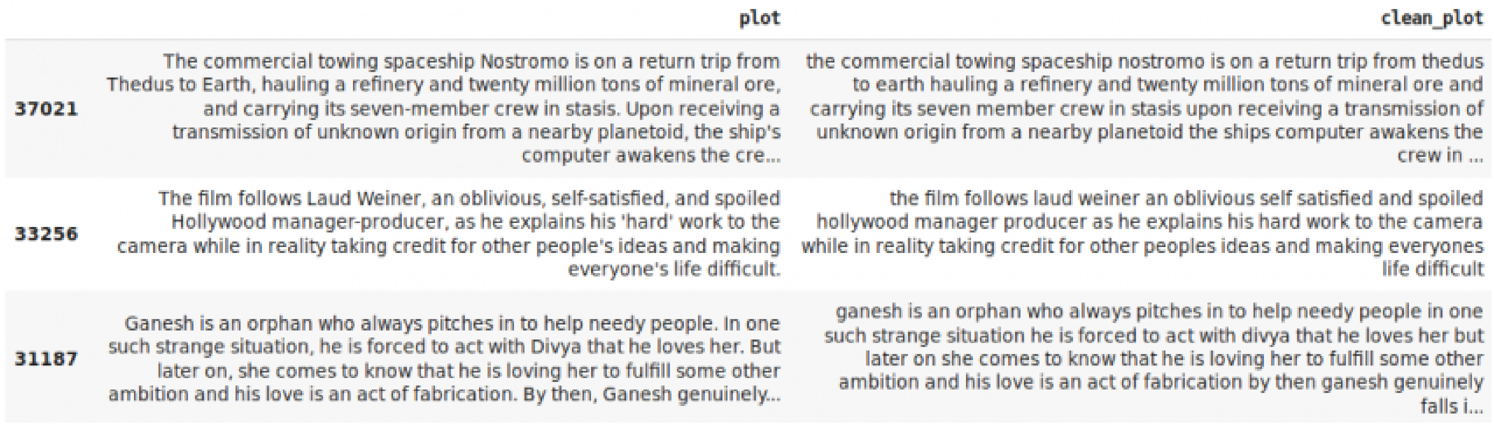
view raw

genre_text_cleaning.py hosted with ❤ by GitHub

Let’s apply the function on the movie plots by using the apply-lambda duo:

```
movies_new['clean_plot'] = movies_new['plot'].apply(lambda x: clean_text(x))
```

Feel free to check the new versus old movie plots. I have provided a few random samples below:



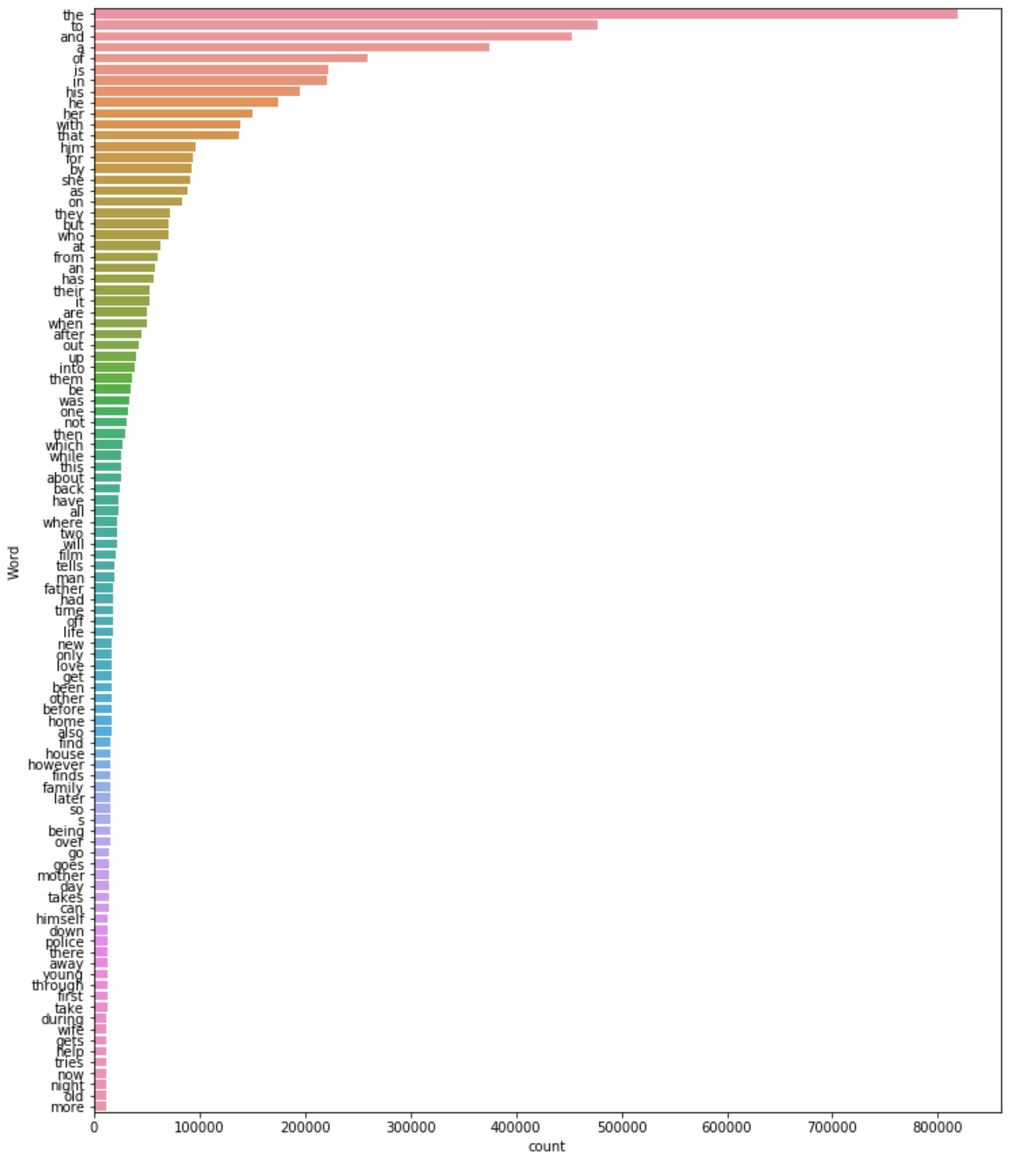
	plot	clean_plot
37021	The commercial towing spaceship Nostromo is on a return trip from Thedus to Earth, hauling a refinery and twenty million tons of mineral ore, and carrying its seven-member crew in stasis. Upon receiving a transmission of unknown origin from a nearby planetoid, the ship's computer awakens the cre...	the commercial towing spaceship nostromo is on a return trip from thedus to earth hauling a refinery and twenty million tons of mineral ore and carrying its seven member crew in stasis upon receiving a transmission of unknown origin from a nearby planetoid the ships computer awakens the crew in ...
33256	The film follows Laud Weiner, an oblivious, self-satisfied, and spoiled Hollywood manager-producer, as he explains his 'hard' work to the camera while in reality taking credit for other people's ideas and making everyone's life difficult.	the film follows laud weiner an oblivious self satisfied and spoiled hollywood manager producer as he explains his hard work to the camera while in reality taking credit for other peoples ideas and making everyones life difficult
31187	Ganesh is an orphan who always pitches in to help needy people. In one such strange situation, he is forced to act with Divya that he loves her. But later on, she comes to know that he is loving her to fulfill some other ambition and his love is an act of fabrication. By then, Ganesh genuinely...	ganesh is an orphan who always pitches in to help needy people in one such strange situation he is forced to act with divya that he loves her but later on she comes to know that he is loving her to fulfill some other ambition and his love is an act of fabrication by then ganesh genuinely falls i...

In the *clean_plot* column, all the text is in lowercase and there are also no punctuation marks. Our text cleaning has worked like a charm.

The function below will visualize the words and their frequency in a set of documents. Let's use it to find out the most frequent words in the movie plots column:

```python
def freq_words(x, terms = 30):
  all_words = ' '.join([text for text in x])
  all_words = all_words.split()
  fdist = nltk.FreqDist(all_words)
  words_df = pd.DataFrame({'word':list(fdist.keys()), 'count':list(fdist.values())})

  # selecting top 20 most frequent words
  d = words_df.nlargest(columns="count", n = terms)

  # visualize words and frequencies
  plt.figure(figsize=(12,15))
  ax = sns.barplot(data=d, x= "count", y = "word")
  ax.set(ylabel = 'Word')
  plt.show()

# print 100 most frequent words
freq_words(movies_new['clean_plot'], 100)
```

view raw
genre_words_visual.py hosted with ❤ by **GitHub**



Most of the terms in the above plot are stopwords. These stopwords carry far less meaning than other keywords in the text (they just add noise to the data). I'm going to go ahead and remove them from the plots' text. You can download the list of stopwords from the *nltk* library:

```
nltk.download('stopwords')
```

Let's remove the stopwords:

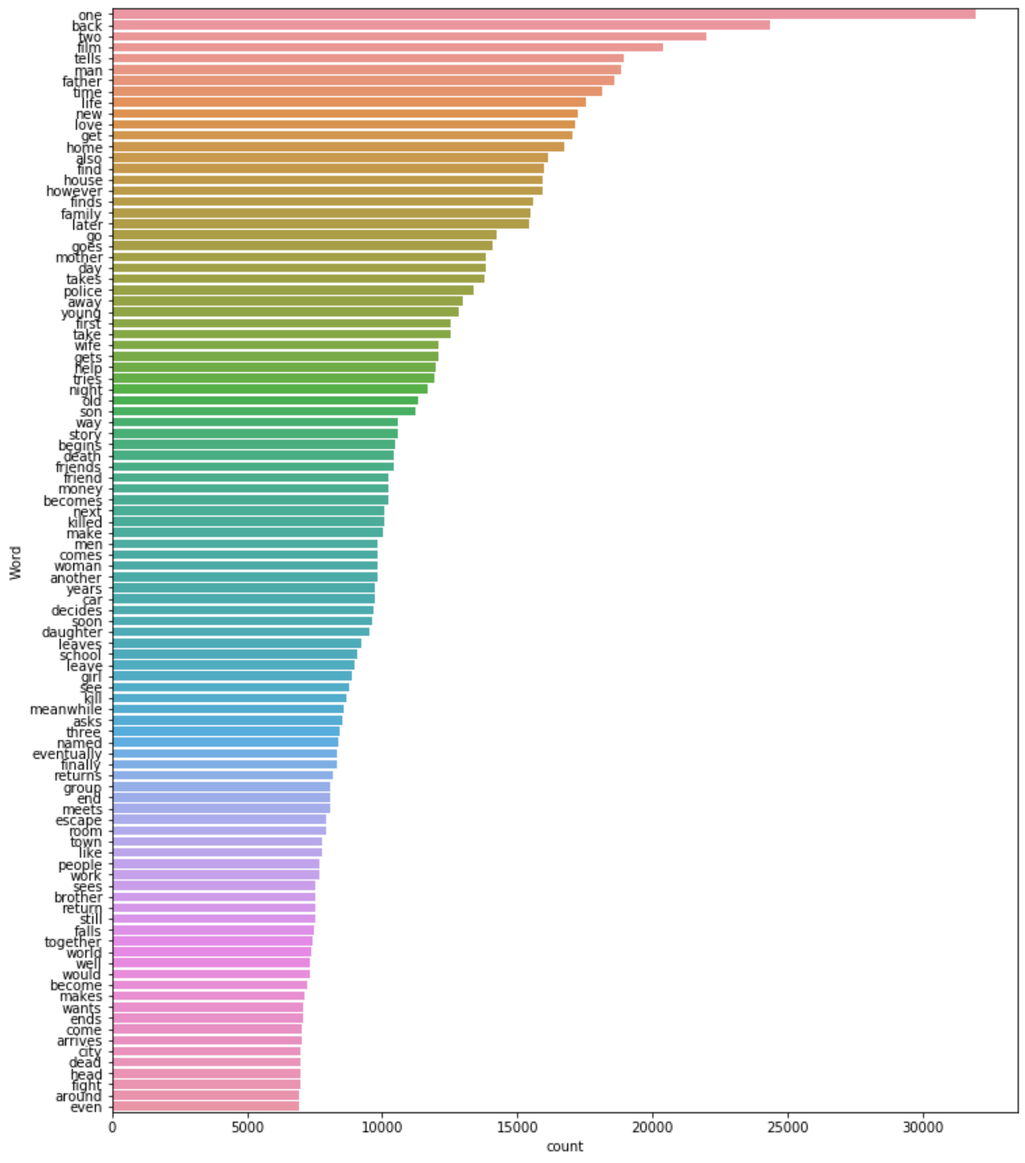
```python
from nltk.corpus import stopwords
stop_words = set(stopwords.words('english'))

# function to remove stopwords
def remove_stopwords(text):
    no_stopword_text = [w for w in text.split() if not w in stop_words]
    return ' '.join(no_stopword_text)

movies_new['clean_plot'] = movies_new['clean_plot'].apply(lambda x: remove_stopwords(x))
```

view raw

genre_stopwords_remove.py hosted with ❤ by **GitHub**

Check the most frequent terms sans the stopwords:

```
freq_words(movies_new['clean_plot'], 100)
```



Looks much better, doesn't it? Far more interesting and meaningful words have now emerged, such as "police", "family", "money", "city", etc.

Converting Text to Features

I mentioned earlier that we will treat this multi-label classification problem as a Binary Relevance problem. Hence, we will now one hot encode the target variable, i.e., *genre_new* by using sklearn's **MultiLabelBinarizer( )**. Since there are 363 unique genre tags, there are going to be 363 new target variables.

```python
from sklearn.preprocessing import MultiLabelBinarizer

multilabel_binarizer = MultiLabelBinarizer()
multilabel_binarizer.fit(movies_new['genre_new'])

# transform target variable
y = multilabel_binarizer.transform(movies_new['genre_new'])
```

view raw

genre_ohe_target.py hosted with ❤ by **GitHub**

Now, it's time to turn our focus to extracting features from the cleaned version of the movie plots data. For this article, I will be using TF-IDF features. Feel free to use any other feature extraction method you are comfortable with, such as Bag-of-Words, word2vec, GloVe, or ELMo.

I recommend checking out the below articles to learn more about the different ways of creating features from text:

- An Intuitive Understanding of Word Embeddings: From Count Vectors to Word2Vec
- A Step-by-Step NLP Guide to Learn ELMo for Extracting Features from Text

```
tfidf_vectorizer = TfidfVectorizer(max_df=0.8, max_features=10000)
```

I have used the 10,000 most frequent words in the data as my features. You can try any other number as well for the *max_features* parameter.

Now, before creating TF-IDF features, we will split our data into train and validation sets for training and evaluating our model's performance. I'm going with a 80-20 split – 80% of the data samples in the train set and the rest in the validation set:

```
# split dataset into training and validation set
xtrain, xval, ytrain, yval = train_test_split(movies_new['clean_plot'], y, test_size=0.2, random_state=9)
```

view raw
genre_traintest_split.py hosted with ❤ by GitHub

Now we can create features for the train and the validation set:

```
# create TF-IDF features xtrain_tfidf = tfidf_vectorizer.fit_transform(xtrain) xval_tfidf = tfidf_vectorizer.transform(xval)
```

Build Your Movie Genre Prediction Model

We are all set for the model building part! This is what we've been waiting for.

Remember, we will have to build a model for every one-hot encoded target variable. Since we have 363 target variables, we will have to fit 363 different models with the same set of predictors (TF-IDF features).

As you can imagine, training 363 models can take a considerable amount of time on a modest system. Hence, I will build a Logistic Regression model as it is quick to train on limited computational power:

```
from sklearn.linear_model import LogisticRegression # Binary Relevance from sklearn.multiclass import OneVsRestClassifier # Performance metric from sklearn.metrics import f1_score
```

We will use sk-learn's *OneVsRestClassifier* class to solve this problem as a Binary Relevance or one-vs-all problem:

```
lr = LogisticRegression() clf = OneVsRestClassifier(lr)
```

Finally, fit the model on the train set:

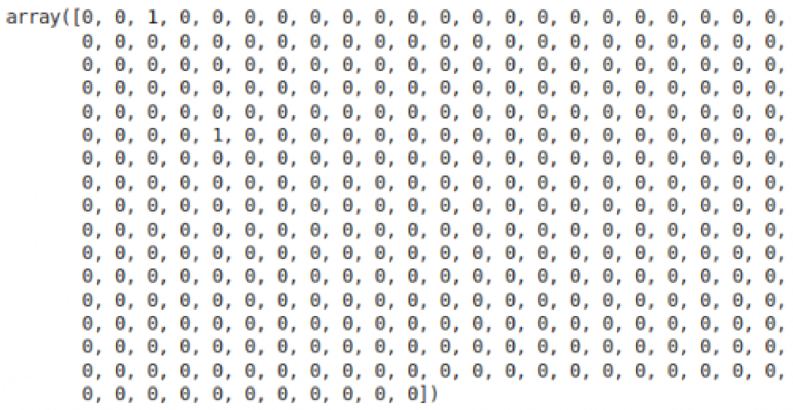
```
# fit model on train data clf.fit(xtrain_tfidf, ytrain)
```

Predict movie genres on the validation set:

```
# make predictions for validation set y_pred = clf.predict(xval_tfidf)
```

Let’s check out a sample from these predictions:

```
y_pred[3]
```



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

It is a binary one-dimensional array of length 363. Basically, it is the one-hot encoded form of the unique genre tags. We will have to find a way to convert it into movie genre tags.

Luckily, sk-learn comes to our rescue once again. We will use the **inverse_transform( )** function along with the *MultiLabelBinarizer( )* object to convert the predicted arrays into movie genre tags:

```
multilabel_binarizer.inverse_transform(y_pred)[3]
```

Output:

```
('Action', 'Drama')
```

Wow! That was smooth.

However, to evaluate our model’s overall performance, we need to take into consideration all the predictions and the entire target variable of the validation set:

```
# evaluate performance f1_score(yval, y_pred, average="micro")
```

Output:

```
0.31539641943734015
```

We get a decent F1 score of 0.315. These predictions were made based on a threshold value of 0.5, which means that the probabilities greater than or equal to 0.5 were converted to 1’s and the rest to 0’s.

Let’s try to change this threshold value and see if that improves our model’s score:

```
# predict probabilities y_pred_prob = clf.predict_proba(xval_tfidf)
```

Now set a threshold value:

```
t = 0.3 # threshold value y_pred_new = (y_pred_prob >= t).astype(int)
```

I have tried 0.3 as the threshold value. You should try other values as well. Let’s check the F1 score again on these new predictions.

```
# evaluate performance f1_score(yval, y_pred_new, average="micro")
```

Output:

```
0.4378456703198025
```

That is quite a big boost in our model’s performance. A better approach to find the right threshold value would be to use a k-fold cross validation setup and try different values.

Create Inference Function

Wait – we are not done with the problem yet. We also have to take care of the new data or new movie plots that will come in the future, right? Our movie genre prediction system should be able to take a movie plot in raw form as input and generate its genre tag(s).

To achieve this, let’s build an inference function. It will take a movie plot text and follow the below steps:

- Clean the text
- Remove stopwords from the cleaned text
- Extract features from the text
- Make predictions
- Return the predicted movie genre tags

```python
def infer_tags(q):
    q = clean_text(q)
    q = remove_stopwords(q)
    q_vec = tfidf_vectorizer.transform([q])
    q_pred = clf.predict(q_vec)
    return multilabel_binarizer.inverse_transform(q_pred)
```

view raw

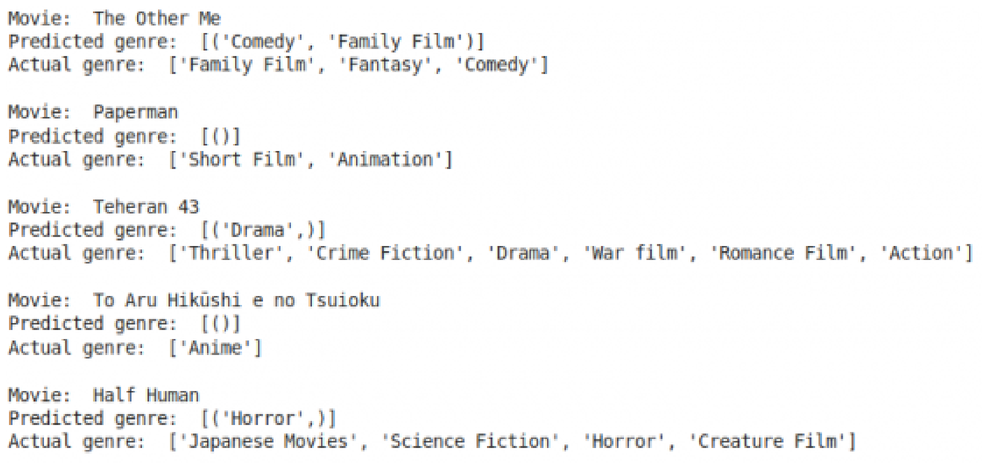
genre_inference.py hosted with ❤ by **GitHub**

Let’s test this inference function on a few samples from our validation set:

```python
for i in range(5):
  k = xval.sample(1).index[0]
  print("Movie: ", movies_new['movie_name'][k], "\nPredicted genre: ", infer_tags(xval[k])), print("Actual genre: ",movies_new['g
```

view raw

genre_infer_newdata.py hosted with ❤ by **GitHub**



```
Movie:  The Other Me
Predicted genre:  [('Comedy', 'Family Film')]
Actual genre:  ['Family Film', 'Fantasy', 'Comedy']

Movie:  Paperman
Predicted genre:  [()]
Actual genre:  ['Short Film', 'Animation']

Movie:  Teheran 43
Predicted genre:  [('Drama',)]
Actual genre:  ['Thriller', 'Crime Fiction', 'Drama', 'War film', 'Romance Film', 'Action']

Movie:  To Aru Hikūshi e no Tsuioku
Predicted genre:  [()]
Actual genre:  ['Anime']

Movie:  Half Human
Predicted genre:  [('Horror',)]
Actual genre:  ['Japanese Movies', 'Science Fiction', 'Horror', 'Creature Film']
```

Yay! We've built a very serviceable model. The model is not yet able to predict rare genre tags but that's a challenge for another time (or you could take it up and let us know the approach you followed).

Where to go from here?

If you are looking for similar challenges, you'll find the below links useful. I have solved a **Stackoverflow Questions Tag Prediction** problem using both machine learning and deep learning models in our course on Natural Language Processing.

The links to the course are below for your reference:

- Certified Course: Natural Language Processing (NLP) using Python
- Certified Program: NLP for Beginners
- The Ultimate AI & ML BlackBelt Program

End Notes

I would love to see different approaches and techniques from our community to achieve better results. Try to use different feature extraction methods, build different models, fine-tune those models, etc. There are so many things that you can try. Don't stop yourself here – go on and experiment!

Feel free to discuss and comment in the comment section below. The full code is available here.

Article Url - https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/

Prateek Joshi

Data Scientist at Analytics Vidhya with multidisciplinary academic background. Experienced in machine learning, NLP, graphs & networks. Passionate about learning and applying data science to solve real world problems.