# ELEC 400M Final Project: Multi-Label Genre Classification of Movies Using NLP

**Andrew Munro-West**
Department of Electrical Engineering
University of British Columbia
Vancouver, BC V6T 1Z4

## Abstract

Multi label genre classification based on movie summary data is a complicated machine learning problem. I chose to do a project exploring the process of cleaning data using natural language processing and how well different machine learning models are able to handle the complex classification problem. My investigations involved comparing logistic regression, random forest, and Multinomial Naive Bayes classifiers. Due to the complexity of the problem the accuracy metrics of my trained models weren't particularly impressive. However, the cleaning process and the exploration of how each model predicted showed interesting results for how each model performed.

## 1 Introduction

For this project I decided to explore models to create a classifier that can predict a movies genre based on the movies plot/synopsis. This genre prediction is a complex multi-label classification problem with Natural language processing(NLP). A movie can have multiple genre tags and genre's are a fairly abstract concept so for the purposes of this project I planned to clean and prep the data and then compare the results obtained through different classifiers.

This type of NLP can be very useful for predictive algorithms to classify media content. If algorithms are able to classify abstract concepts like genre based on descriptions or summaries with decent accuracy it could help sorting algorithms to suggest like-content to end users.

## 2 The Data set and problem

To accomplish this project I used the CMU Movie Summary Corpus available for free at

`http://www.cs.cmu.edu/~ark/personas/`

this dataset contains 42,306 movie plot summaries extracted from Wikipedia and movie metadata matched to the appropriate Wikipedia movie id.

Before cleaning the data it needed to be processed into a workable format. I began by loading the meta data file into a pandas dataframe, this was simple because it came in a table format already. The summary data started as a text document which I read into memory and split into it's summary and Wikipedia id components, which I then loaded into a dataframe. I renamed the columns of my 2 dataframes appropriately and then merged the dataframes to create a dataset containing the movie id, movie name, genre tags, and plot summary. The genre tags were initially in a string format but I converted them to a workable list format and then set out to clean the genre tags.

### 2.1 Cleaning the Data

#### 2.1.1 The Genre label cleaning

initially, in my data set of 43000 movies there were about 15000 unique combinations of genre tags. After changing all strings to lowercase the total number of individual genres was lowered to 363 unique genre tags. upon inspection of these tags I found that the majority of these tags were either; a spelling error, a combination of other tags, a tag that says nothing about the content of the movie or an obscure sub-genre that could be decomposed into 1 or more of the more common genres. More than 90% of these tags appeared in the dataset 1-10 times where my "main" genres would occur 1,000-20,000 times.

I created a look up table and cleaning algorithm based on my impressions and descriptions of the various genres and sorted through the genre data removing and/or replacing all of the troublesome tags. Therebye, reducing my initial 363 unique genre tags down to 30 tags with a minimum number of occurrences of 106. (Code snippets can be found in Appendix*)

#### 2.1.2 Synopsis NLP cleaning

The synopsis data needed to be cleaned using basic NLP. First I removed all non-alphanumeric characters from each synopsis in my dataset reducing the synopsis' down to just words and spaces. The synopsis data could now be broken down and individual word counts could be used as input features for my models.

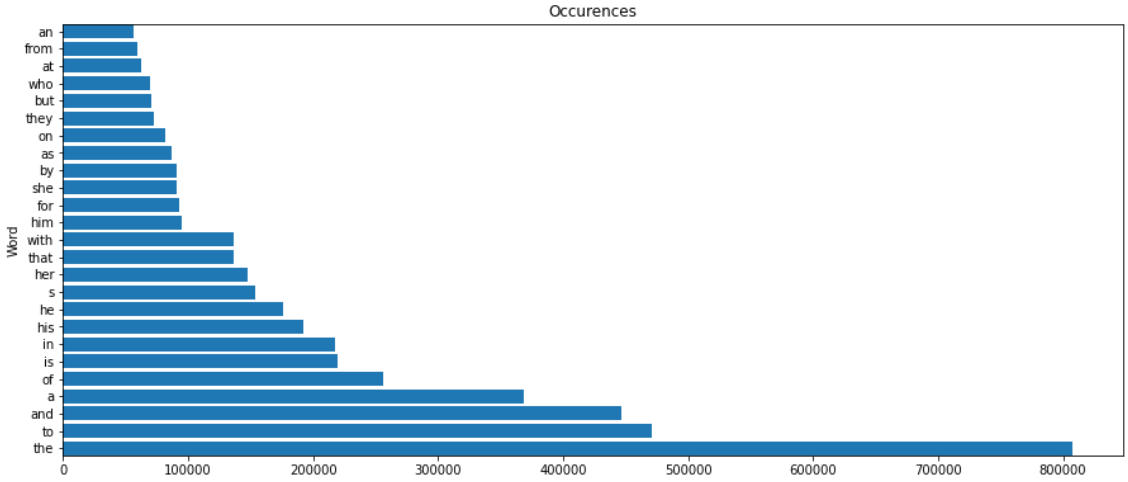


Figure 1: Word distribution with stop words.

Figure 1 shows the top 25 words that dominated my dataset. We can observe that the dataset was dominated by words like "the", "and". "in". These words are known as stop words in English and are grammar words which hold no information about the subject of the writing. For our purposes, these words add no useful information to our dataset and essentially pollute our features with noise. To get rid of these words I imported a list of stop words from the nltk python package and created a function that iterates through each synopsis removing the stop words.

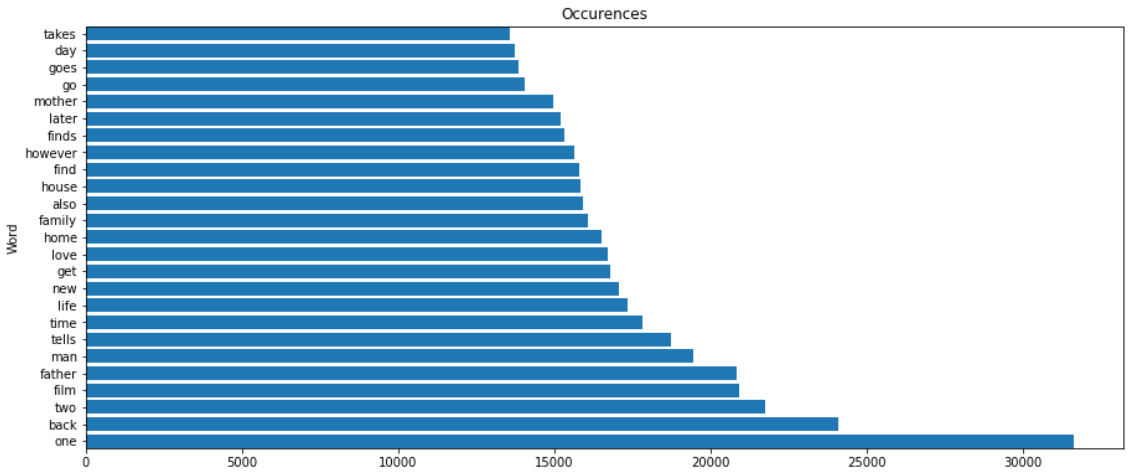


Figure 2: Word distribution without stop words.

Figure 2 shows the synopsis dataset after removing stop words. We can see that the distribution is quite a bit more balanced and words more meaningful to the individual synopsis' show up like "love", and "life".

### 2.2 Feature engineering and data splitting

Once the data had been adequately cleaned, I needed to transform the word synopsis data into meaningful word counts to use in my algorithms. To accomplish this I used sklearn's TfidfVectorizer. TfidfVectorizer essentially takes a word count across multiple documents and measures how often each word was used in a particular document against how often it was used in all of the documents. This essentially gives higher weight to keywords that appear in specific documents a lot and reduces the weight of common words such as stop words which are common across many documents. Using this vectorizer I converted my synopsis data into weights associated with each word giving the model training data that should help it determine genres based on frequency of certain keywords.

An alternative to TfidfVectorizer was also explored with CountVectorizer. CountVectorizer is a much simpler algorithm which essentially turns a list of words into a vector of word counts. CountVectorizer puts no weight on any individual words based on frequency and focuses on the word counts as features.

For my genre data I was left with a list of 30 classes that my models would need to predict into. To make this more workable for my models I used sklearn's multilabelBinarizer to transform the 30 labels into an array of 30 binary values.

```
('crime', 'horror', 'mystery', 'thriller')
[0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0]
```

Figure 3: MultilabelBinarizer transform.

## 3 Classifiers

Three classifiers were observed, logistic regression, random forest, and Multinomial Naive Bayes. Due to memory limitations, Five-fold cross validation and hyper-parameter tuning was applied only to the logistic regression algorithm.

### 3.1 Logistic Regression

Logistic regression classification was chosen because it's a relatively simple algorithm for classification and the one I am the most familiar with. Logistic regression typically works as a binary classification so in this application I put the logistic regression model through an sklearn OneVsRestClassifier to create my estimator. A OneVsRestClassifier essentially takes a binary classifier model and creates a classifier that runs through all of the binary classifications in our output labels. So my Logistic regression classifier can actually be thought of as 30 separate binary logistic regression models which predict whether or not each of my 30 labels are present. I subjected this model to 5 fold cross validation and used a gridsearchCV to tune the C hyper parameter for values between 0.01 and 1000. Given the complexity of the gridsearchCV training took approximately 13 minutes.

### 3.2 Random Forest

Random forest classification was chosen as an alternate approach to logistic regression, Random forest is a method of ensemble learning that generates decision trees for classification which vote on what label to apply to a particular class. Random forest inherently supports multi-label classification and I chose to build the classifier with 100 trees using the Shannon entropy criterion for tree splitting. For the random forest classifier I also scaled the tfidf data to unit std zero mean. This criteria essentially is used to minimize the log-loss between the predictions and the true labels of the individual trees. In this configuration training a single model took approximately 7 minutes.

$$H(Q_m) = -\sum_k p_{mk} \log(p_{mk}) \tag{1}$$

### 3.3 Multinomial Naive Bayes

Finally I tried using Multinomial Naive Bayes classification. Multinomial Naive Bayes comes highly recommended for NLP classification problems due to the it's lightweight approach of measuring the probability a document belongs in a particular class based on the frequency of which terms appear in the document. For this approach I found that using CountVectorizer instead of TfidfVectorizer to simply count the number of occurrences of the top 10,000 terms led to better prediction accuracy. Training this model took 0.5 seconds which is extremely lightweight in comparison to the other two models.

### 3.4 Scoring and Results

Simple accuracy scores check the number of correct perfectly correct classifications against the number of incorrect classifications. If a model were to get a single label wrong for a sample then it would be counted as an incorrect classification. Because I am dealing with a multiclass problem with 30 different labels for each model to predict to, simple accuracy scores don't provide much information about how accurate my models are.

Because of this, I decided to use the f1-score as my accuracy metric. The f1-score is a measure of harmonic mean of the precision and recall.

$$F1 = 2 * (precision * recall)/(precision + recall) \tag{2}$$

The f1-score I used applied 'micro' averaging which calculates the score by counting the total true positives, false negatives and false positives globally across all labels.

With this scoring I found the Logistic regression algorithm to be the most accurate of the 3 getting a score of 0.51 2nd best was Multinomial Naive Bayes with a score of 0.44 and last was random forest with a score of 0.23.

```
Optimal model:
lr: OneVsRestClassifier(estimator=LogisticRegression(C=10.0, random_state=42,
                                                     solver='sag'))
F1 scores based on micro average:
logistic regression:  0.516852230553662
random forest: 0.2353065390300556
multnomial Naive Bayes:  0.4443030653556969
```

Figure 4: Result scores.

Putting the predicted classes and the actual classes into a table we can make some observations about why the scores are as low as they are.

| | Actual | Logistic Regression | Random forest | Multinomial Naive Bayes |
|---|---|---|---|---|
| **8125** | (action,) | () | () | (drama,) |
| **8126** | (crime, drama, fiction, historical) | (drama, fiction, thriller) | (drama,) | (drama, fiction) |
| **8127** | (drama, musical, romance) | (drama,) | (drama,) | (drama, romance) |
| **8128** | (comedy, drama, indie) | (drama,) | (drama,) | (drama,) |
| **8129** | (drama,) | (drama,) | (drama,) | (drama,) |
| **8130** | (drama,) | (action, drama) | (drama,) | (drama,) |
| **8131** | (drama,) | (drama,) | () | () |
| **8132** | (action, adventure, fiction, sci-fi) | (action, adventure, drama) | (drama,) | (action, adventure, fiction, sci-fi) |
| **8133** | (short,) | (musical, romance) | (drama,) | () |
| **8134** | (cult, fiction, horror, sci-fi) | (fiction, horror, sci-fi) | () | (fiction, sci-fi) |
| **8135** | (comedy, drama) | (comedy,) | () | (comedy, drama) |

Figure 5: Prediction table.

## 4 Discussion and Conclusions

We can see from Figure 5 that our random forest classifier seems much more likely to predict no genre than my other 2 models, It also seems to be particularly biased to picking the more common genres of drama and comedy over other genres. The Naive Bayes model was a lot less afraid to make mistakes and was willing to predict multiple genres with more frequency, and the Logistic regression model performed the best overall managing to get 1 at least 1 correct label in most of its predictions.

Part of the reason that my classifiers seem to be so inclined to predict no genre despite the training set containing no movies with no genre is due to the OneVsRest classifier that I passed my models through. The OneVsRest classifier looks at the data as 30 individual binary classifications to create the multilabel classification we want to see. However, because the models are looking at binary classifications of whether or not to include a particular tag, they have no motivation to pick any tag if they don't meet the individual classification criteria.

Due to the complexity of this problem and the number of labels I was optimistic to see a classification accuracy as high as 50%. I believe that this project has served as a nice introduction into the field of NLP for machine learning and if I were to continue working on this project I would experiment with

improving accuracy by simplifying the classification down to 3 or 4 classes and fixing the issue of no classification before increasing the complexity of the dataset again. I would also try to balance the dataset a bit more finding more movies which fulfill less common tags to hopefully reduce the bias seen by the random forest classifier.

## References


Agarwal, Anjali. "Movie Genre Classifier Using Natural Language Processing." Medium, Medium, 7 Aug. 2020, https://medium.com/@aagarwal691/movie-genre-classifier-using-natural-language-processing-4d5e73da7b78.

Hilsdorf, Max. "Dealing with List Values in Pandas Dataframes." Medium, Towards Data Science, 6 Sept. 2020, https://towardsdatascience.com/dealing-with-list-values-in-pandas-dataframes-a177e534f173.

Jallouli, AMINE. "Genre Classification Based on Wiki Movies Plots." Kaggle, Kaggle, 10 Dec. 2018, https://www.kaggle.com/code/aminejallouli/genre-classification-based-on-wiki-movies-plots/notebook3.-Classifiers-Training.

Joshi, Prateek. "Movie Genre Prediction Using Multi Label Classification." Analytics Vidhya, 19 July 2022, https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/h2$_8$.

Li, Susan. "Multi Label Text Classification with Scikit-Learn." Medium, Towards Data Science, 23 Apr. 2018, https://towardsdatascience.com/multi-label-text-classification-with-scikit-learn-30714b7819c5.

Schroeder, Adam. "Countvectorizer, TfidfVectorizer, Predict Comments." Kaggle, Kaggle, 23 Feb. 2018, https://www.kaggle.com/code/adamschroeder/countvectorizer-tfidfvectorizer-predict-comments/notebookCountVectorizer—-Brief-Tutorial.

# 5 Appendix

```
# read the summaries text and split it into 2 lists for movie id and synopsis
with open("/content/gdrive/MyDrive/data/imdb_movies_dataset/plot_summaries.txt", 'r') as f:
  lines = f.readlines()

for line in lines:
  id.append(line.split('\t')[0])
  synopsis.append(line.split('\t')[1])

# add the id and synopsis to our large dataset
dataset_42306 = pd.DataFrame({'movie_id': id, 'synopsis': synopsis})
# convert the id from int to string
metadata_df['movie_id'] = metadata_df['movie_id'].astype(str)
# merge the 2 dataframes based on movie_id
dataset_42306 = pd.merge(dataset_42306, metadata_df[['movie_id', 'movie_name', 'genre']], on = 'movie_id')
dataset_42306['genre'] = dataset_42306['genre'].str.lower()
# convert the values of our genre column from str to dict to list of values
dataset_42306['genre'] = dataset_42306['genre'].apply(lambda x: list(json.loads(x).values()))
# look at previous cells for custom made genre-cleaning algorithm
dataset_42306['genre'] = clean_genres(dataset_42306['genre'])
# remove entries that have no genre
dataset_41796 = dataset_42306[~(dataset_42306['genre'].str.len() == 0)]
```

```
def clean_genres(series):
  # spelling corrections and special genres splits
  genres = series.apply(lambda y: list(map(lambda x: x.replace("comdedy", "comedy"), y)) )

  genres = genres.apply(lambda y: list(map(lambda x: x.replace("japaneses", "japanese"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("docudrama", "drama documentary"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("film noir", "crime drama"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("chase", "action adventure"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("gladiators", "action historical"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("illnesses & disabilities", "educational documentary"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("b-western", "b-movie western"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("giallo", "crime mystery"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("splatter", "horror crime"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("apocalyptic and post-apocalyptic", "sci-fi"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("rockumentary", "musical documentary"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("tragicomedy", "tragedy comedy"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("future noir", "science crime drama"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("neo-noir", "crime drama"), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("comedy-drama", "comedy drama"), y)) )
  # prefix and suffix removals
  genres = genres.apply(lambda y: list(map(lambda x: x.replace(" movie", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace(" film", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace(" cinema", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace(" effort", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace(" picture", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("nuclear ", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("gulf ", ""), y)) )
  genres = genres.apply(lambda y: list(map(lambda x: x.replace("therimin ", ""), y)) )
```

```
# genre lookup table sets

main_genre_LUT = {
    'action':{'doomsday', 'disaster', 'superhero', 'revenge'},
    'adult' : {'softcore porn', 'softcore pornography', 'hardcore pornography', 'erotica', 'erotic', 'sexploitation', 'pornography', 'pornographic','homoeroticism', 'sex', 'pinku eiga', 'porn'},
    'adventure':{'jungle', 'epic', 'swashbucklers', 'travel', 'road'},
    'animation':{'anime', 'silhouette animation', 'supermarionation','clay', 'cartoon', 'stop motion', 'computer animation'},
    'b-movie':{},
    'black-and-white':{'goat gland', 'silent'},
    'comedy':{'stand-up', 'beach party', 'beach', 'comedy of manners', 'comedy of errors', 'satire', 'buddy', 'humour', 'comedies', 'stoner', 'slapstick', 'screwball', 'parody'},
    'crime':{'addiction', 'juvenile delinquency', 'criminals', 'law & crime', 'women in prisons', 'biker', 'legal', 'detective', 'cop', 'prison', 'outlaw', 'heist', 'courtroom', 'gangster', 'caper', 'escape'},
    'documentary':{'documentaries', 'documetary', 'hagiography', 'nature', 'mockumentary', 'mondo', 'blography', 'biographical', 'biopic [feature]', 'film à clef', 'film & television history'},
    'drama':{'slice of life', 'slice of life story', 'interpersonal relationships', 'melodrama', 'tragedy', 'coming-of-age','coming of age', 'medical'},
    'educational':{'archaeology', 'essay', 'language & literature', 'media studies', 'news', 'environmental science', 'archives and records', 'libraries and librarians', 'graphic & applied arts', 'computers',
    'family':{"children\'s issues', 'childhood', 'family & personal relationships', 'children\'s', 'teen', 'holiday', 'school story', 'family-oriented'},
    'fantasy':{'fantasies', 'sword and sorcery', 'sword and sorcerys', 'fairy tale', 'revisionist fairy tale'},
    'fiction':{'dystopia', 'fictional'},
    'historical':{'mythological', 'period piece','history','period', 'sword and sandal' },
    'holiday':{'christmas'},
    'horror':{ 'road-horror', 'psycho-biddy', 'demonic child', 'alien invasion', 'scary', 'slasher', 'monster','zombie', 'alien', 'creature', 'demonic', 'z', 'horrors'},
    'indie':{'avant-garde', 'dogme 95', 'expressionism', 'mumblecore', 'absurdism', 'experimental', 'surrealism', 'existentialism', 'gross out', 'gross-out', 'art', 'fan'},
    'lgbt':{'gender issues', 'gay', 'gay interest', 'gay themed', 'new queer'},
    'musical':{'singing', 'dance', 'punk rock', 'concert', 'opera', 'instrumental music', 'operetta', 'music', 'breakdance', 'hip hops', 'hip hop', 'film-opera', 'singing '},
    'mystery':{'whodunit', 'conspiracy'},
    'political':{'black', 'social problem', 'americana', 'blaxploitation', 'media', 'albino bias', 'exploitation', 'early black', 'race', 'patriotic', 'feminist', 'social issues', 'neorealism', 'journalism',
    'religious':{'heavenly', 'christian'},
    'romance':{'chick flick', 'romantic'},
    'sci-fi':{'time travel', 'steampunk', 'sci fi originals', 'sci fi pictures originals', 'science', 'space', 'cyberpunk'},
    'short':{'movie serial', 'anthology', 'television'},
    'sports':{'racing', 'auto racing', 'horse racing', 'parkour in popular culture', 'bruceploitation', 'health & fitness', 'baseball', 'boxing', 'martial arts'},
    'spy':{'glamorized spy'},
    'supernatural':{'werewolf','haunted house', 'vampires'},
    'suspense':{'kafkaesque'},
    'thriller':{'thrillers','plague', 'psychological'},
    'war':{'combats', 'cavalry', 'foreign legion', 'warfare', 'private military company', 'the netherlands in world war ii'},
    'western':{'spaghetti western', 'cowboy', 'hybrid western', 'revisionist western'},
    'world':{'domestic', 'new hollywood', 'tamil', 'tollywood', 'british empire', 'indian', 'british new wave', 'malayalam', 'bengali', 'northern', 'czechoslovak new wave', 'latino', 'tokusatsu', 'bollywood',
    }
```

| index ▼ | Actual | Logistic Regression | Random forest | Multinomial Naive Bayes |
|---|---|---|---|---|
| 8177 | drama | action,adventure | | action,adventure |
| 8176 | action,adventure,thriller | | drama | drama |
| 8175 | comedy | comedy | | comedy |
| 8174 | documentary,drama,war | drama | drama | drama,war |
| 8173 | comedy,drama,romance | drama,romance | drama | drama |
| 8172 | comedy,drama,political,romance | comedy,drama | drama | drama |
| 8171 | drama | drama | drama | comedy,drama |
| 8170 | horror | | | |
| 8169 | drama | drama | drama | drama |
| 8168 | animation,comedy | short | comedy | short |
| 8167 | drama,historical,political,thriller | action,thriller | drama | action,thriller |
| 8166 | war | comedy | | |
| 8165 | drama,romance,short | drama | drama | drama |
| 8164 | crime,drama,fiction,historical,mystery,thriller | crime,drama,fiction,mystery,thriller | drama | crime,fiction,mystery,thriller |
| 8163 | comedy,drama,family,romance,sports | comedy,romance | | comedy |
| 8162 | action,adventure,drama,western | action,adventure,western | | action,adventure,western |
| 8161 | drama | action,adventure,thriller | | action,adventure |
| 8160 | drama,romance | drama,romance | drama | drama |
| 8159 | adventure,drama,family,fantasy,romance | drama,horror | | drama,horror |
| 8158 | comedy,fiction,sci-fi,short | animation,comedy,family,fantasy,sci-fi,short | | animation,comedy,family,short |
| 8157 | horror | crime,drama,fiction,horror,mystery,thriller | drama | thriller |
| 8156 | documentary | | | drama |
| 8155 | action,fiction,horror,sci-fi,thriller | action,fiction,sci-fi,thriller | | action,fiction |
| 8154 | animation | adventure,animation,family | | adventure,family |
| 8153 | drama | drama | | drama |
| 8152 | drama,horror | horror,thriller | | |
| 8151 | horror,thriller | drama,thriller | drama | drama |
| 8150 | action,adventure,short | adventure,family | | adventure |
| 8149 | action,adventure,comedy,crime,drama,fiction,indie,political | action,crime | | drama |
| 8148 | comedy | comedy,family,musical | | drama |