Predicting Movie Genres Based on Plot Summaries

Group No: 10

Participants List:

1. Sri Lalana
2. Parsis Presswala
3. Trapti Khandelwal

Problem Statement: This model can used to classify the genre based on the movie plot which helps the audience to make choices accordingly.

Abstract:

This project explores several Machine Learning methods to predict movie genres based on plot summaries. Genres tell us what to expect from the movie. Naive Bayes, Logistic Regression, Adapted Algorithm and Neural Language Processing (NLP) are used for text classification. Experiments with more than 5500 movies, we have perform classification and learned approach achieves the best result on the test set. Our task is to build a model that can predict the genre of a movie using just the plot details (available in text form).

Multi-Label Classification:

In machine learning, multi-label classification and the strongly related problem of multi-output classification are variants of the classification problem where multiple labels may be assigned to each instance.

In multi-label classification, the training set is composed of instances each associated with a set of labels, and the task is to predict the label sets of unseen instances through analysing training instances with known label sets. Difference between multi-class classification & multi-label classification is that in multi-class problems the classes are mutually exclusive, whereas for multi-label problems each label represents a different classification task, but the tasks are somehow related.

Here, in the example, 'X' represents the input variables and 'y' represents the target variables:

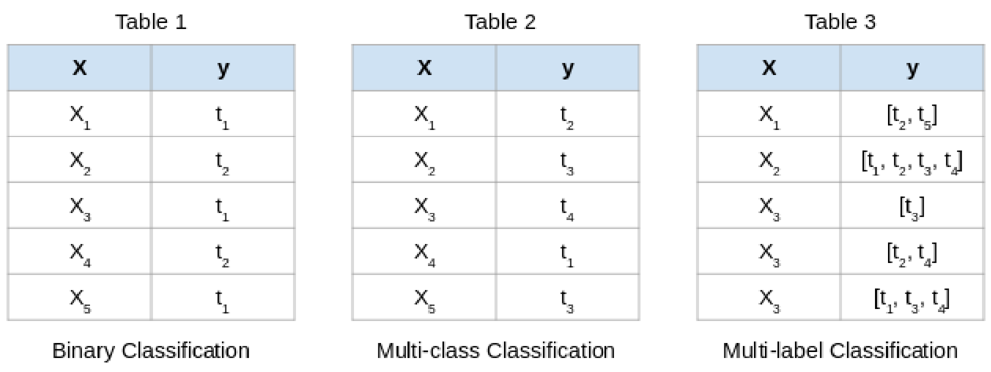


Table 1

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

About the Dataset

- **movies_project_final.xls**
- **Fields:** Title, Genre, Synopsis, Year, Duration, Caste, Rating, Language
    - Here we have more than 5500 data

Implementation

Import the required libraries

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

warnings.filterwarnings("ignore")
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')

pd.set_option('display.max_colwidth', 300)
```

Load Data

```
df = pd.read_excel("movies_project_final.xls")
df.head(5)
```

	Title	Genre	Synopsis	Year	Duration	Caste	Rating	Language
0	The Shawshank Redemption	Drama	Two imprisoned men bond over a number of years...	1994	142 min	Tim Robbins, Morgan Freeman, Bob Gunton, Willi...	9.3	English
1	The Dark Knight	Action, Crime, Drama	When the menace known as the Joker wreaks havo...	2008	152 min	Christian Bale, Heath Ledger, Aaron Eckhart, M...	9.0	English
2	Inception	Action, Adventure, Sci-Fi	A thief who steals corporate secrets through t...	2010	148 min	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen...	8.8	English
3	Fight Club	Drama	An insomniac office worker and a devil-may-car...	1999	139 min	Brad Pitt, Edward Norton, Meat Loaf, Zach Grenier	8.8	English
4	Pulp Fiction	Crime, Drama	The lives of two mob hitmen, a boxer, a gangst...	1994	154 min	John Travolta, Uma Thurman, Samuel L. Jackson,...	8.9	English

Handling the missing values

```
df.isnull().sum()
```

```
Title         0
Genre       953
Synopsis      2
Year        596
Duration    596
Caste       600
Rating      596
Language    596
dtype: int64
```

```
df = df[pd.notnull(df['Synopsis'])]
df = df[pd.notnull(df['Genre'])]
df.isnull().sum()
```

This code helps us to extract all the genres from the movies data:

```
# extract genres
genres = []
for i in df['Genre']:
   s = [x.strip() for x in i.split(',')]
   genres.append(s)

df['genre_new'] = genres
df['genre_new']
# remove samples with 0 genre tags
df_new = df[~(df['Genre'].str.len() == 0)]
df_new.shape, df.shape
```

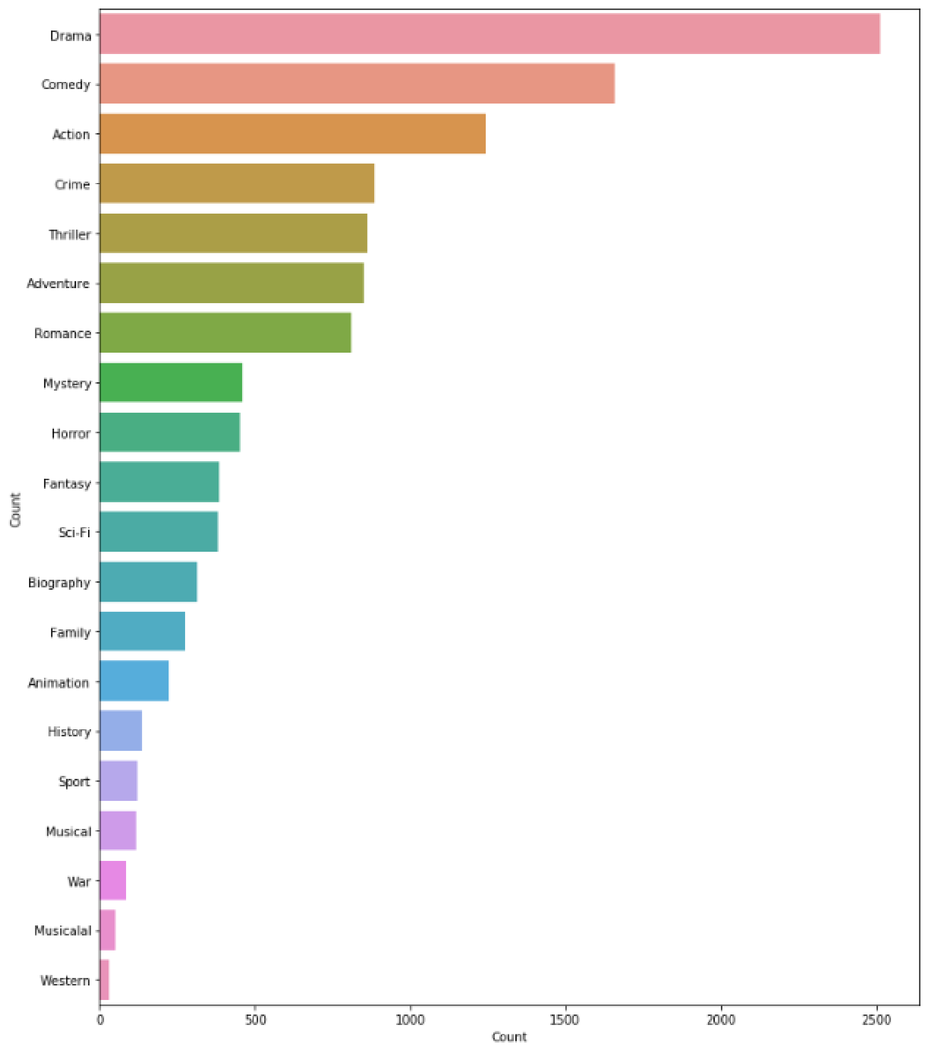
Output: ((4641, 9), (4641, 9))

```
# add to 'movies' dataframe
all_genres = sum(df['genre_new'],[])
len(set(all_genres))
```

Output: 19

There are over 19 unique genre tags in our dataset. We will use FreqDist( ) from the nltk (Natural Language Toolkit) library to create a dictionary of genres and their occurrence count across the dataset:

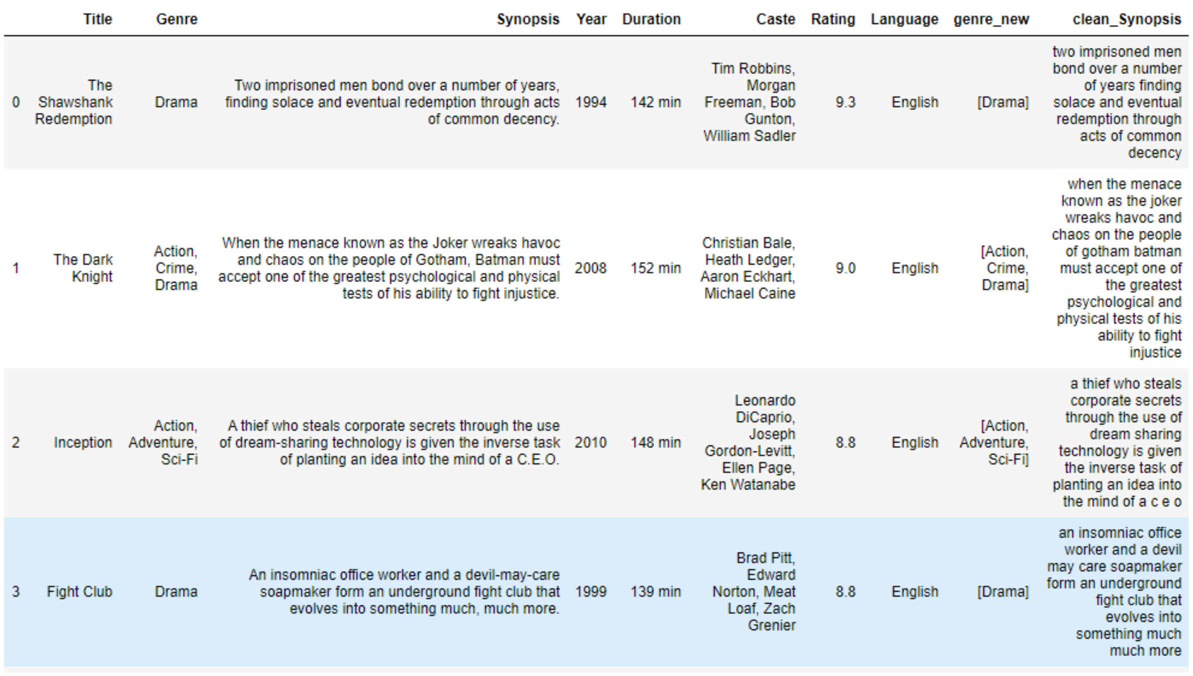
```
all_genres = nltk.FreqDist(all_genres)

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

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()
```



It will clean our data a bit. In the *clean Synopsis* column, all the text is in lowercase and there are also no punctuation marks.

```
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
```

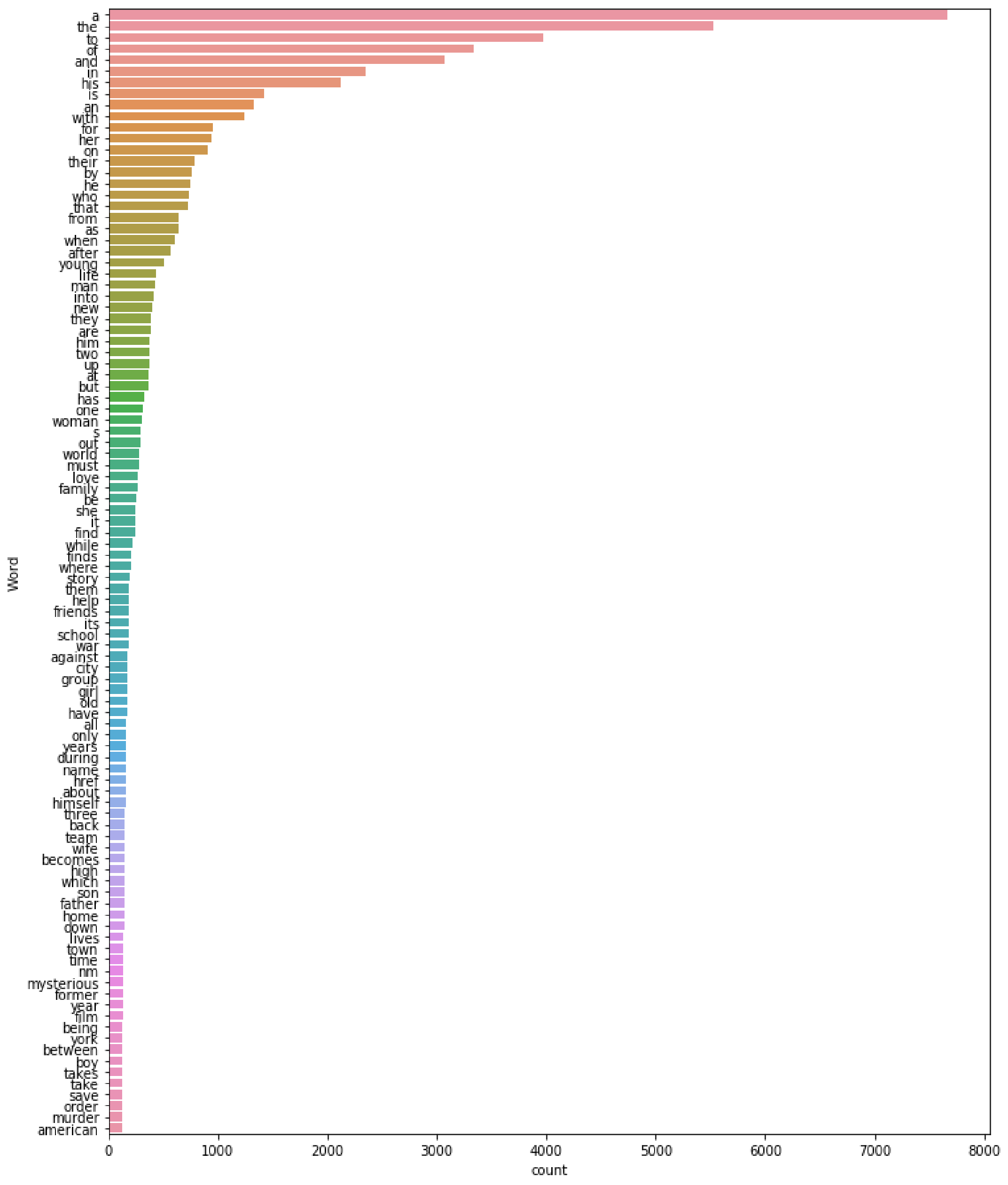
```
df_new['clean_Synopsis'] = df_new['Synopsis'].apply(lambda x: clean_text(x))
df_new.head()
```



	Title	Genre	Synopsis	Year	Duration	Caste	Rating	Language	genre_new	clean_Synopsis
0	The Shawshank Redemption	Drama	Two imprisoned men bond over a number of years, finding solace and eventual redemption through acts of common decency.	1994	142 min	Tim Robbins, Morgan Freeman, Bob Gunton, William Sadler	9.3	English	[Drama]	two imprisoned men bond over a number of years finding solace and eventual redemption through acts of common decency
1	The Dark Knight	Action, Crime, Drama	When the menace known as the Joker wreaks havoc and chaos on the people of Gotham, Batman must accept one of the greatest psychological and physical tests of his ability to fight injustice.	2008	152 min	Christian Bale, Heath Ledger, Aaron Eckhart, Michael Caine	9.0	English	[Action, Crime, Drama]	when the menace known as the joker wreaks havoc and chaos on the people of gotham batman must accept one of the greatest psychological and physical tests of his ability to fight injustice
2	Inception	Action, Adventure, Sci-Fi	A thief who steals corporate secrets through the use of dream-sharing technology is given the inverse task of planting an idea into the mind of a C.E.O.	2010	148 min	Leonardo DiCaprio, Joseph Gordon-Levitt, Ellen Page, Ken Watanabe	8.8	English	[Action, Adventure, Sci-Fi]	a thief who steals corporate secrets through the use of dream sharing technology is given the inverse task of planting an idea into the mind of a c e o
3	Fight Club	Drama	An insomniac office worker and a devil-may-care soapmaker form an underground fight club that evolves into something much, much more.	1999	139 min	Brad Pitt, Edward Norton, Meat Loaf, Zach Grenier	8.8	English	[Drama]	an insomniac office worker and a devil may care soapmaker form an underground fight club that evolves into something much much more

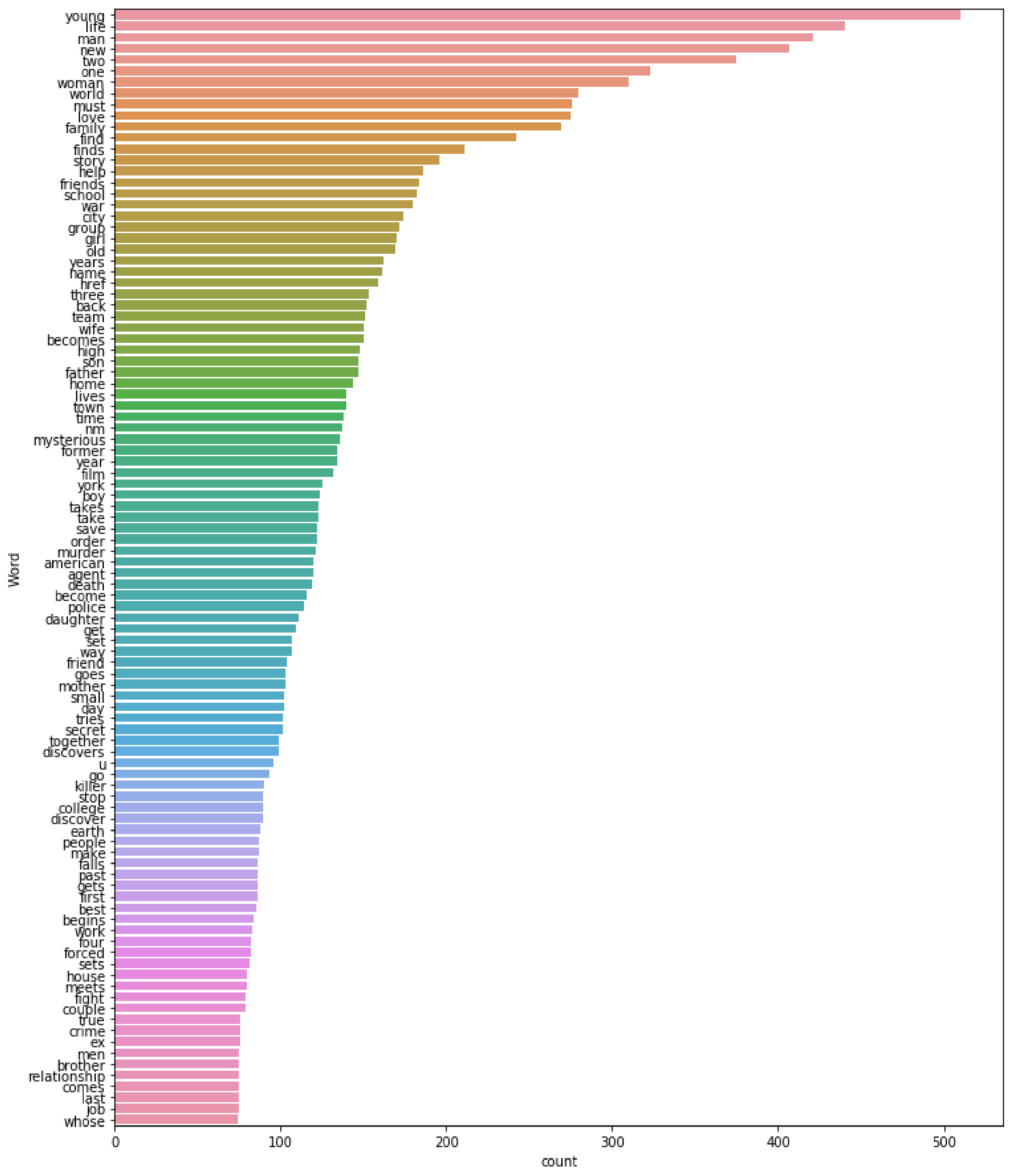
The function below will visualize the words and their frequency in a set of documents.

```
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(df_new['clean_Synopsis'], 100)
```



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).

```
nltk.download('stopwords')

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)

df_new['clean_Synopsis'] = df_new['clean_Synopsis'].apply(lambda x: remove_stopwords(x))

freq_words(df_new['clean_Synopsis'], 100)
```



Converting Text to Features

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 20 unique genre tags, there are going to be 20 new target variables.

```
from sklearn.preprocessing import MultiLabelBinarizer

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

# transform target variable
y = multilabel_binarizer.transform(df_new['genre_new'])
#len(y[0])
```

```
multilabel_binarizer.classes_
```

Extract features from the cleaned version of the movie plots data. Here we will be using TF-IDF features (other methods are Bag-of-Words, word2vec, GloVe, or ELMo).

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

Splitting training and testing

```
from sklearn.model_selection import train_test_split
# split dataset into training and validation set
xtrain, xval, ytrain, yval = train_test_split(df_new['clean_Synopsis'], y, test_size=0.2, random_state=1
0)
```

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)
```

USING LOGISTIC REGRESSION

- One-vs-the-rest, this strategy consists in fitting one classifier per class, the class is fitted against all the other classes. Advantage of this approach is its interpretability. Since each class is represented by one classifier only, it is possible to gain knowledge about the class by inspecting its corresponding classifier.

- In One-vs-the-rest strategy, one could build multiple independent classifiers and, for an unseen instance, choose the class for which the confidence is maximized.

- The main assumption here is that the labels are *mutually exclusive*. You do not consider any underlying correlation between the classes in this method.

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

```
lr = LogisticRegression(penalty='l2', random_state=10,max_iter=1000,class_weight='balanced')
clf_logistic = OneVsRestClassifier(lr)
```

# Fitting a model

```
clf_logistic.fit(xtrain_tfidf, ytrain)
```

# Make a prediction

```
y_pred_logistic = clf_logistic.predict(xval_tfidf)
```

# Evaluate performance

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_recall_fscore_support
print ("F1 Score is:    ",f1_score(yval, y_pred_logistic, average="micro"))
print ("The Accuracy is:",accuracy_score(yval,y_pred_logistic))
print ()
p, r, f, s = precision_recall_fscore_support(yval, y_pred_logistic,average="micro",warn_for=('precisio
n', 'recall', 'f-score'))
print ("f beta score    ",f)
print ("Precison:       ",p)
print ("Recall:         ",r)
```

```
F1 Score is:     0.5829909613804437
The Accuracy is: 0.11194833153928956

f beta score     0.5829909613804437
Precison:        0.5689655172413793
Recall:          0.5977253580454929
```

CONCLUSION: We are using logistic regression with One vs Rest method.As it is multilabel classification, penalty='l1' is not supported, So we have to used penalty 'l2' or 'None'. With the penalty 'l2', we found better model. We have imbalance classes in our dataset, so we have applied class_weight. After applying logistic regression, we found accuracy with 11% and F1 Score with 58% amd also we got Precison 56.8% and recall 59.7%.

LOGISTIC REGRESSION WITH KFOLD

```
from sklearn.model_selection import cross_val_score
print ("Logistic: ")
model1 = cross_val_score(clf_logistic,xtrain_tfidf, ytrain,scoring="accuracy",cv=15)
accuracy = model1.mean() * 100
print ("Acccuracy of Logistic is: ",accuracy)
```

```
Logistic:
Acccuracy of Logistic is:  12.176657524705064
```

CONCLUSION: We tried to apply Cross validation on the previous Logistic model and we found 12.04% accuracy which is 1% more than previous model.

USING DECISION TREE

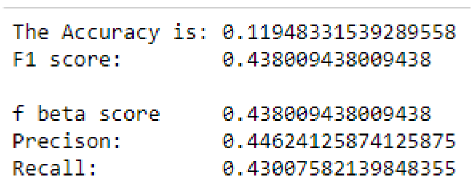
```
from sklearn.tree import DecisionTreeClassifier
clf_decision_tree = DecisionTreeClassifier(criterion='gini',splitter='best',random_state=10)
clf_decision_tree.fit(xtrain_tfidf,ytrain)
```

# Make a prediction

```
y_pred_decision_tree = clf_decision_tree.predict(xval_tfidf)
```

# Evaluate performance

```
print ("The Accuracy is:",accuracy_score(yval,y_pred_decision_tree))
print ("F1 score:      ",f1_score(yval, y_pred_decision_tree, average="micro"))
print ()
p, r, f, s = precision_recall_fscore_support(yval, y_pred_decision_tree,average="micro",warn_for=('p
recision', 'recall', 'f-score'))
print ("f beta score    ",f)
print ("Precison:       ",p)
print ("Recall:         ",r)
```



```
The Accuracy is: 0.11948331539289558
F1 score:        0.438009438009438

f beta score     0.438009438009438
Precison:        0.44624125874125875
Recall:          0.43007582139848355
```

CONCLUSION: After applying Logistic Regression, we tried Decision tree classifier. In which we have used gini criterion with random splitter. From which we found same accuracy as Logistic Regression with Cross Validation 12%, but our f1 score falls slightly to 46.7%. Here we found 47.6% Precison while recall is 45.9%

USING BINARY RELEVANCE

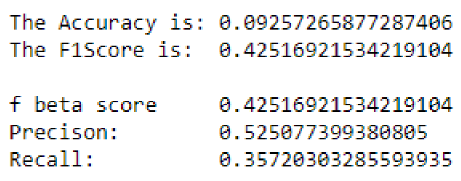
- In this case an ensemble of single-label binary classifiers is trained, one for each class. Each classifier predicts either the membership or the non-membership of one class. The union of all classes that were predicted is taken as the multi-label output. This approach is popular because it is easy to implement, however it also ignores the possible correlations between class labels. In other words, if there's *q* labels, the binary relevance method create *q* new data sets, one for each label and train single-label classifiers on each new data set.

- See the state of the art of binary relevance from three perspectives. First, basic settings for multi-label learning and binary relevance solutions are briefly summarized. Second, representative strategies to provide binary relevance with label correlation exploitation abilities are discussed. Third, some of our recent studies on binary relevance aimed at issues other than label correlation exploitation are introduced. As a conclusion, we provide suggestions on future research directions.

- One-vs-the-rest and Binary Relevance seem very much alike. If multiple classifiers in One-vs-the-rest answer *"yes"* then you are back to the binary relevance scenario.

```
# using binary relevance
from skmultilearn.problem_transform import BinaryRelevance
from sklearn.naive_bayes import GaussianNB

# initialize binary relevance multi-label classifier
# with a gaussian naive bayes base classifier
clf1 = BinaryRelevance(GaussianNB())

# train
y_pred1 = clf1.fit(xtrain_tfidf, ytrain)

# predict
predictions1 = clf1.predict(xval_tfidf)

#print(y_pred1)
#print(predictions1)
```

```
print("The Accuracy is:",accuracy_score(yval,predictions1))
# evaluate performance
print("The F1Score is: ",f1_score(yval, predictions1,average="micro"))
print ()
p, r, f, s = precision_recall_fscore_support(yval, predictions1,average="micro",warn_for=('precision',
'recall', 'f-score'))
print ("f beta score    ",f)
print ("Precison:       ",p)
print ("Recall:         ",r)
```



```
The Accuracy is: 0.09257265877287406
The F1Score is:  0.42516921534219104

f beta score     0.42516921534219104
Precison:        0.525077399380805
Recall:          0.35720303285593935
```

CONCLUSION: We are using BinaryRelevance from problem_transform. In which we are using naive_bayes base claassifier. From which we got 9.2% accuracy, 42.5% f1 score, 52.5% Precision and 35.7% Recall.

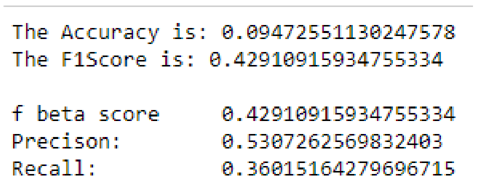
USING CLASSIFIER CHAINS

- A chain of binary classifiers C0, C1,…,Cn is constructed, where a classifier Ci uses the predictions of all the classifier Cj , where j < i. This way the method, also called classifier chains (CC), can take into account label correlations.

- The total number of classifiers needed for this approach is equal to the number of classes, but the training of the classifiers is more involved.

```
from skmultilearn.problem_transform import ClassifierChain
from sklearn.naive_bayes import GaussianNB

# initialize classifier chains multi-label classifier
# with a gaussian naive bayes base classifier
clf2 = ClassifierChain(GaussianNB())

# train
y_pred2=clf2.fit(xtrain_tfidf, ytrain)

# predict
predictions2 = clf2.predict(xval_tfidf)
```

```
print("The Accuracy is:",accuracy_score(yval,predictions2))
# evaluate performance
print("The F1Score is:",f1_score(yval, predictions2,average="micro"))
print ()
p, r, f, s = precision_recall_fscore_support(yval, predictions2,average="micro",warn_for=('precision',
'recall', 'f-score'))
print ("f beta score     ",f)
print ("Precison:       ",p)
print ("Recall:        ",r)
```



```
The Accuracy is: 0.09472551130247578
The F1Score is: 0.42910915934755334

f beta score      0.42910915934755334
Precison:         0.5307262569832403
Recall:           0.36015164279696715
```

<u>CONCLUSION:</u> We are using ClassifierChain from problem_transform with naive_bayes. From which we got 9.4% accuracy, 42.9% f1 score, 53% Precision and 36% Recall. Here we got almost same result as Binary relevance.

USING LABEL POWERSET

- This approach does take possible correlations between class labels into account. More commonly this approach is called the label-power set method, because it considers each member of the power set of labels in the training set as a single label.

- This method needs worst case (2^|C|) classifiers, and has a high computational complexity.

- However when the number of classes increases the number of distinct label combinations can grow exponentially. This easily leads to combinatorial explosion and thus computational infeasibility. Furthermore, some label combinations will have very few positive examples.

```
from skmultilearn.problem_transform import LabelPowerset
from sklearn.naive_bayes import GaussianNB

# initialize Label Powerset multi-label classifier
# with a gaussian naive bayes base classifier
clf3 = LabelPowerset(GaussianNB())

# train
clf3.fit(xtrain_tfidf, ytrain)

# predict
predictions3 = clf3.predict(xval_tfidf)
```

```
print("The Accuracy is:",accuracy_score(yval,predictions3))
print("The F1Score is: ",f1_score(yval, predictions3,average="micro"))
print ()
p, r, f, s = precision_recall_fscore_support(yval, predictions3,average="micro",warn_for=('precision',
'recall', 'f-score'))
print ("f beta score    ",f)
print ("Precison:       ",p)
print ("Recall:         ",r)
```

```
The Accuracy is: 0.17115177610333693
The F1Score is:  0.521077792264233

f beta score     0.521077792264233
Precison:        0.5381508078994613
Recall:          0.5050547598989048
```

CONCLUSION: We are using here another class Label Powerset from problem_transform with naive_bayes. From which we got 17% accuracy, 52% f1 score, 53.8% Precision and 50.5% Recall. Among all the methods we used from problem_transform, Label Powerset provides the best result. Even we got the highest accuracy(17.1%) among all the classification method.

ADAPTED ALGORITHM

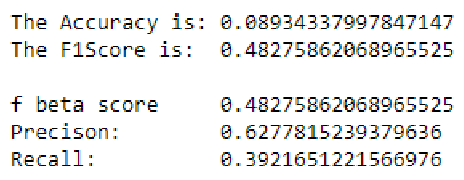
- Algorithm adaptation methods for multi-label classification concentrate on adapting single-label classification algorithms to the multi-label case usually by changes in cost/decision functions.

- Here we use a multi-label learning approach named ML-KNN which is derived from the traditional K-nearest neighbour (KNN) algorithm.

```
from skmultilearn.adapt import MLkNN

classifier1 = MLkNN(k=20)

# train
classifier1.fit(xtrain_tfidf, ytrain)
```

```
# predict
predictions = classifier1.predict(xval_tfidf)
```

```
print("The Accuracy is:",accuracy_score(yval,predictions))
print("The F1Score is: ",f1_score(yval, predictions,average="micro"))
print ()
p, r, f, s = precision_recall_fscore_support(yval, predictions,average="micro",warn_for=('precision', 'recall
', 'f-score'))
print ("f beta score    ",f)
print ("Precison:       ",p)
print ("Recall:         ",r)
```



```
The Accuracy is: 0.08934337997847147
The F1Score is:  0.48275862068965525

f beta score     0.48275862068965525
Precison:        0.6277815239379636
Recall:          0.3921651221566976
```

CONCLUSION: We are using another method called Adapted Algorithm which basically will perform Multiple K Nearest Neighbour algorithm. From which we got 8.9% accuracy, 48.2% f1 score, 62.7% Precision and 39.2% Recall.

FINAL CONCLUSION:

We have applied different approaches using different methods. Here we got maximum accuracy from label power set with 17%, but we can not consider it in multilabel classification. So we will evaluate our model with macro and micro F1 Score, Precision and Recall. We got maximum F1 score from Logistic Regression with One vs Rest method with 58%. So we got best model from Logistic Regression with One vs Rest method.