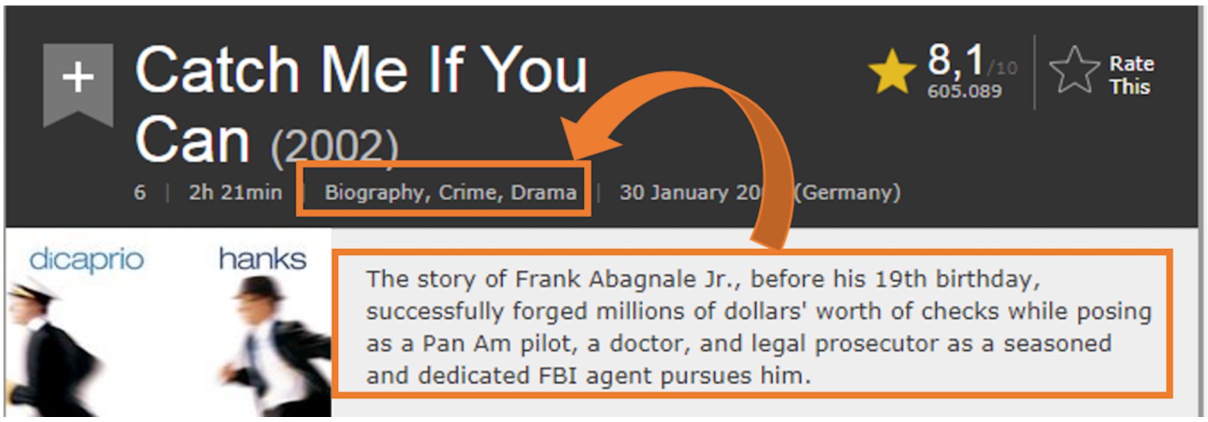
MULTI-LABEL CLASSIFICATION

Akhil Raju Pusapati

1.Introduction: *2*
2.Data Setup *2*
2.1 Data Load **2**
2.2 Data Pre-Processing **2**
2.2.1 Clean Genre 3
2.1.1.1 One-Hot Encoding Genre's 3
2.1.1.2 Visualize Genre's 4
2.2.2 Clean Summary 5
2.2.2.1 Visualize Summary 6
2.3 Split Data Into Train, Test Set and Validation Set. **7**
3. Data Transformation *8*
4. Training The Classifier *9*
4.1 Build The Classifier **10**
4.2 Train and Predict the outcomes **10**
5. Evaluation *11*
5.1 Hyper Parameter Tuning **12**
6. Conclusion *13*
6.1 Best Model **13**

1.Introduction:

This task aims to train a book dataset based on description and build a classifier so that future book descriptions can be classified to specific genre's from the built classifier model. I am using CMU book dataset made available by Carnegie Mellon University. This dataset contains plot summaries for 16,559 books extracted from Wikipedia, along with aligned metadata from Freebase, including book author, title, and genre. Performance evaluation is performed on the linear models.
As I am doing text based classification, I will be pre-processing the text and will evaluate the performance of multiple models to choose the model with the best accuracy.


Catch Me If You Can (2002)
8,1/10
605.089
Rate This
6 | 2h 21min | Biography, Crime, Drama | 30 January 20 (Germany)
dicaprio
hanks

The story of Frank Abagnale Jr., before his 19th birthday, successfully forged millions of dollars' worth of checks while posing as a Pan Am pilot, a doctor, and legal prosecutor as a seasoned and dedicated FBI agent pursues him.

2.Data Setup

2.1 Data Load

The data provided by CMU is in a text format. It needs to be converted into a data frame that will allow us to perform further analysis. The data frame containing all the information is called "books". It has 16,559 rows and 6 columns.

```
books
```

	book_id	book_name	book_author	book_publishedyear	genre	summary
0	620	Animal Farm	George Orwell	1945-08-17	{"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":...	Old Major, the old boar on the Manor Farm, ca...
1	843	A Clockwork Orange	Anthony Burgess	1962	{"/m/06n90": "Science Fiction", "/m/0l67h": "N...	Alex, a teenager living in near-future Englan...
2	986	The Plague	Albert Camus	1947	{"/m/02m4t": "Existentialism", "/m/02xlf": "Fi...	The text of The Plague is divided into five p...

16559 rows × 6 columns

Before pre-processing the data we can observe that there are total 16,559 rows, each containing a book's name, descriptions and their respective genre's.

2.2 Data Pre-Processing

From the above picture we can observe that, the "books" data frame is not structured and cleaned. Firstly, we can observe that the "genre" column has values in json format. Secondly, the "summary" column has information with unnecessary characters. These columns need to be pre-processed. It will help bring our text into a form that is predictable and analysable for the task of classifying the genre's. Finally, we can observe that few columns like the “published year”, “book author” will not be of much use in predicting the genre. Hence, they need to be dropped.

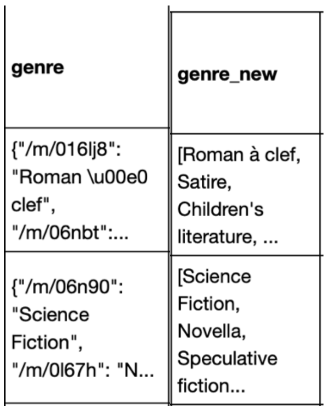
2.2.1 Clean Genre

The first step in pre-processing the “books” data Frame is to drop all the rows with empty genre values. We need to understand that missing values are definitely undesirable but it is difficult to quantify the magnitude of effects in statistical and machine learning projects. If it’s a large dataset and a very small percentage of data is missing the effect may not be detectable at all. However, if the dataset is relatively small, every data point counts. In these situations, a missing data point means loss of valuable information. In any case, generally missing data creates imbalanced observations, cause biased estimates, and in extreme cases, can even lead to invalid conclusions. On removing the empty values the **number of rows has gone down from 16,559 to 12,841**.

```
# The length has gone down from 16,559 to 12,841. Since we removed the rows with empty genre'.
len(books)
```

```
12841
```

The next step is to clean the genre’ column. It is be noticed that, the column is serialized into json format and hence we can use “json.loads()” function to parse the string in the dictionary and de-serialize it. By using this function we can grab the json values in the dictionary and place them in a new list in the “genre_new” column. From here on we can just use the cleaned “genre-new” column for further analysis.



genre	genre_new
{"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":...	[Roman à clef, Satire, Children's literature, ...
{"/m/06n90": "Science Fiction", "/m/0l67h": "N...	[Science Fiction, Novella, Speculative fiction...

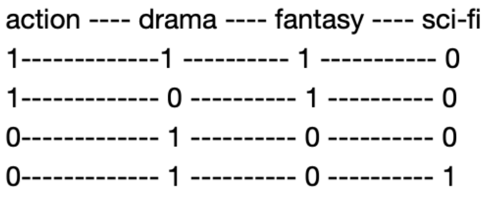
Finally, we can observe that the total number of unique genres are 227. The genre’s will be the labels for their respective description.

```
# Number of unique labels.
all_genres = sum(genres,[])
len(set(all_genres))
```

```
227
```

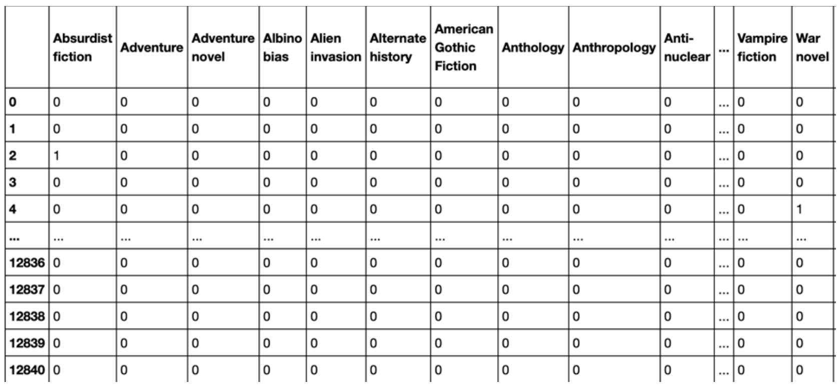
2.1.1.1 One-Hot Encoding Genre’s

Our data is categorical and machine learning algorithms cannot work directly with them. So we have to convert categorical data into numbers that can be provided to machine learning algorithms to do a better job in classifying data. for example, let says we have a simple sequence of “*genre’s*” with the values “*Action*”, “*Drama”, “Fantasy”* and “*Sci-fi”*. We can use integer encoding for the data to represent the values like this “*Action*” = 0, “*Drama*” = 1, “*Fantasy*” = 2 and “*Sci-fi”=3* . But, if we use this encoding and allow the model to assume a natural ordering between categories, it may result in poor performance or unexpected results. Hence, the integer encoded variable is removed and a new binary variable is added for each unique integer value using one hot encoding method. In the “genre’s” sequence example, “1” value is placed in the binary variable for the genre which is detected and “0” values for the genre’s which are not present for each row in the data frame.



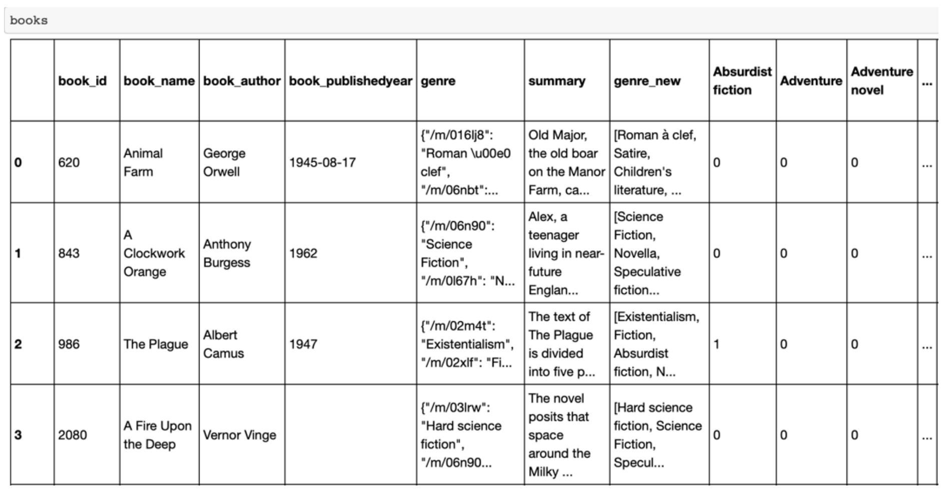
```
action ---- drama ---- fantasy ---- sci-fi
1------------1 ---------- 1 ----------- 0
1------------ 0 ---------- 1 ---------- 0
0------------ 1 ---------- 0 ---------- 0
0------------ 1 ---------- 0 ---------- 1
```

For the books data frame, we use “MultiLabel Binarizer”. Although a list of sets or tuples is a very intuitive format for multilabel data, it is unwieldy to process. This transformer helps to convert between the intuitive format and the supported multilabel format. The inverse transform of the “MultiLabel Binarizer” allows to Transform the given indicator matrix into label sets.



	Absurdist fiction	Adventure	Adventure novel	Albino bias	Alien invasion	Alternate history	American Gothic Fiction	Anthology	Anthropology	Anti-nuclear	...	Vampire fiction	War novel
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
2	1	0	0	0	0	0	0	0	0	0	...	0	0
3	0	0	0	0	0	0	0	0	0	0	...	0	0
4	0	0	0	0	0	0	0	0	0	0	...	0	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
12836	0	0	0	0	0	0	0	0	0	0	...	0	0
12837	0	0	0	0	0	0	0	0	0	0	...	0	0
12838	0	0	0	0	0	0	0	0	0	0	...	0	0
12839	0	0	0	0	0	0	0	0	0	0	...	0	0
12840	0	0	0	0	0	0	0	0	0	0	...	0	0

We can observe that the labels for each row are converted into one hot encoding and are placed in Data frame. This temporary data frame is concatenated with the books data frame which can be used to train and test the model. The final data frame can be seen below.

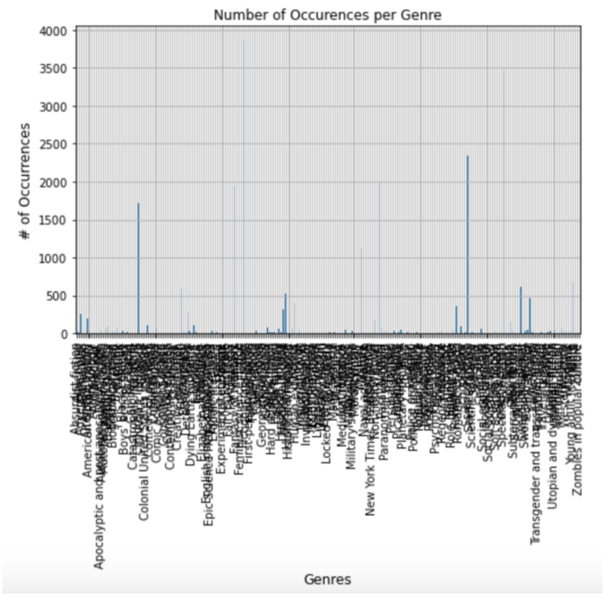


```
books
```

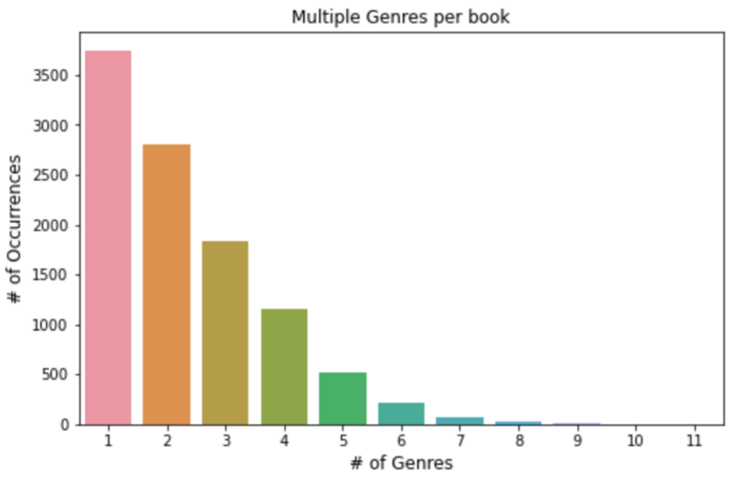
	book_id	book_name	book_author	book_publishedyear	genre	summary	genre_new	Absurdist fiction	Adventure	Adventure novel	...
0	620	Animal Farm	George Orwell	1945-08-17	{"/m/016lj8": "Roman \u00e0 clef", "/m/06nbt":...	Old Major, the old boar on the Manor Farm, ca...	[Roman à clef, Satire, Children's literature, ...	0	0	0	...
1	843	A Clockwork Orange	Anthony Burgess	1962	{"/m/06n90": "Science Fiction", "/m/0l67h": "N...	Alex, a teenager living in near-future Englan...	[Science Fiction, Novella, Speculative fiction...	0	0	0	...
2	986	The Plague	Albert Camus	1947	{"/m/02m4t": "Existentialism", "/m/02xlf": "Fi...	The text of The Plague is divided into five p...	[Existentialism, Fiction, Absurdist fiction, N...	1	0	0	...
3	2080	A Fire Upon the Deep	Vernor Vinge		{"/m/03lrw": "Hard science fiction", "/m/06n90...	The novel posits that space around the Milky ...	[Hard science fiction, Science Fiction, Specul...	0	0	0	...

2.1.1.2 Visualize Genre’s

On cleaning the genre's let us visualize them to have a better idea of the dataset. Firstly, Lets plot the genre' and their occurrences.



This visualization is very bad as there are 227 labels. Hence we cannot make any inferences from it. So, Let us try to make another visualization by counting the number occurrences of genre' per book.



This indeed just revealed something not surprising but important to know.
1) most of our Descriptions have 1-4 genres.
2) No Descriptions has all Genres at a time.

2.2.2 Clean Summary

In cleaning, There is one last column that needs to be cleaned and it is the description column. The description or the summary needs to be cleaned by :

1) Removing wrong characters such as special characters.
2) Need to change the case of all characters that make the comment to avoid case insensitivity our models.
3) Need to remove stop words since those are words that are likely to be common to all comments.

Let us see the uncleaned summary :

```
" The Ravenous introduces us to Eddie Spears, a teenager who is into video games and hanging out with his best friend, Jess Br
own. Eddie has a problem: he can hear distant whispers and this causes severe headaches. When he discovers the true cause for
his town's prosperity: a sacrificial pseudo-Druid cult, Eddie comes to realize he has a special gift—but can he use it in time
to save his sister's life?"
```

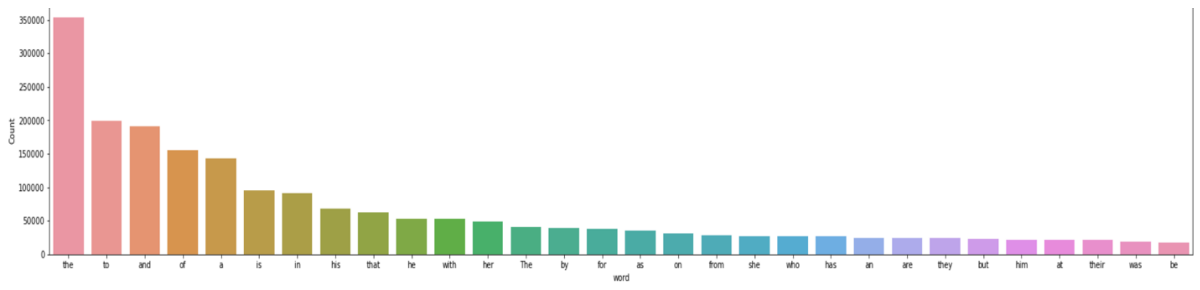
We can observe that it has un-wanted characters, special characters and stop words. The special character' can be removed using regex equations and are replaced with a space. The stop words can be removed using the "NLTK" package which contains the "STOP WORDS".

The cleaned summary will look as follows :

```
'ravenous introduces us eddie spears teenager video games hanging best friend jess brown eddie problem hear distant whispers c
auses severe headaches discovers true cause towns prosperity sacrificial pseudodruid cult eddie comes realize special giftbut
use time save sisters life'
```

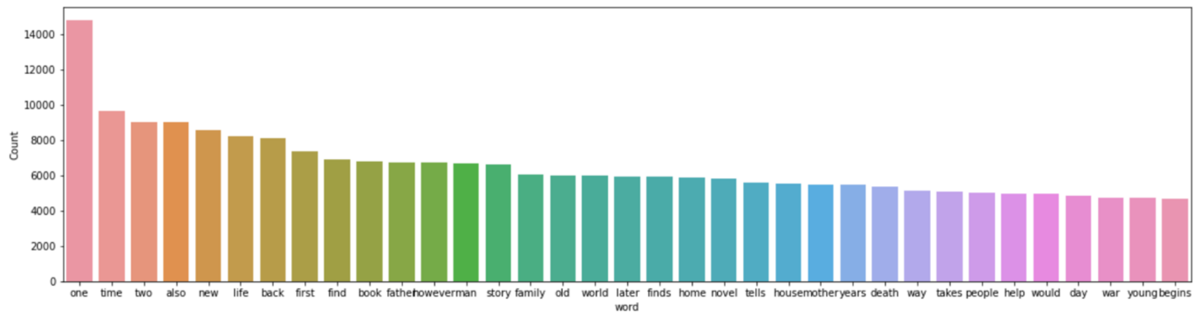
2.2.2.1 Visualize Summary

Firstly, let us first visualize the summary before it is cleaned :



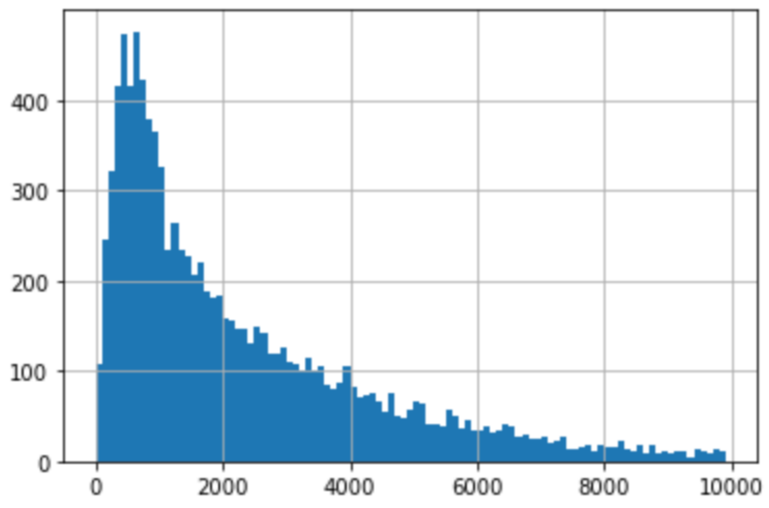
From this Image we can observe that most frequent words in the summary are stop words like " the, and, to, an, who etc." These stop words need to be removed from the given text so that more focus can be given to those words which define the meaning of the text.

Secondly, let us visualize the summary after it is cleaned :



Immediately after cleaning, we can observe the difference in the frequently occurring words. Once the summary is cleaned, stop words are no more the most frequently occurring words. Words like "one", "time", "two", "father", "family" which are more relatable are recurring.

Finally, Let us see the composition of our summary data: let see how big are texts in general :



This graph tells us that max length of the descriptions is in between 500 and 400charecters. The ones with less that 100 characters are very few.

2.3 Split Data Into Train, Test Set and Validation Set.

The train-test split is a technique for evaluating the performance of a machine learning algorithm. It can be used for classification or regression problems and can be used for any supervised learning algorithm. The procedure involves taking a dataset and dividing it into two subsets. The first subset is used to fit the model and is referred to as the training dataset. The second subset is not used to train the model; instead, the input element of the dataset is provided to the model, then predictions are made and compared to the expected values. This second dataset is referred to as the test dataset. Finally there is also an option third subset called the validation set and it is used to tune the hyper parameters to find the best model with a good accuracy and low loss.

Train Dataset: Used to fit the machine learning model.

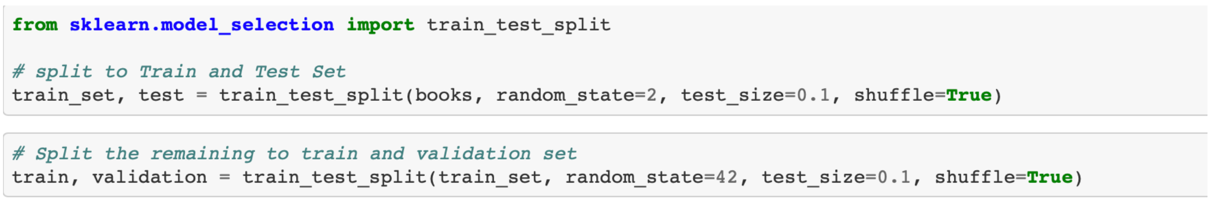
Test Dataset: Used to evaluate the fit machine learning model.

Validation Dataset: Used to find and tune the hyper parameters for the model.

The objective is to estimate the performance of the machine learning model on new data: data not used to train the model. This is how we expect to use the model in practice. Namely, to fit it on available data with known inputs and outputs, then make predictions on new examples in the future where we do not have the expected output or target values. The train-test procedure is appropriate when there is a “sufficiently large dataset” available.

The idea of “sufficiently large” is specific to each predictive modelling problem. It means that there is enough data to split the dataset into train, test and validation datasets and each of the datasets are suitable representations of the problem domain. This requires that the original dataset is also a suitable representation of the problem domain. A suitable representation of the problem domain means that there are enough records to cover all common cases and most uncommon cases in the domain. This might mean combinations of input variables observed in practice. It might require thousands, hundreds of thousands, or millions of examples.

Conversely, the train-test procedure is not appropriate when the dataset available is small. The reason is that when the dataset is split into train, test and validation sets, there will not be enough data in the training dataset for the model to learn an effective mapping of inputs to outputs. There will also not be enough data in the test set to effectively evaluate the model performance. The estimated performance could be overly optimistic (good) or overly pessimistic (bad). Finally, validation set will not have enough features to tune the hyper parameters.



```
from sklearn.model_selection import train_test_split

# split to Train and Test Set
train_set, test = train_test_split(books, random_state=2, test_size=0.1, shuffle=True)
```

```
# Split the remaining to train and validation set
train, validation = train_test_split(train_set, random_state=42, test_size=0.1, shuffle=True)
```

In our case, we can see from the above picture that we have split the data into train, test and validation sets.

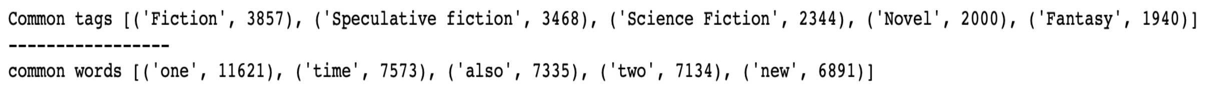
3. Data Transformation

The data is cleaned and hence, we move into the next step which is data transformation. In this section we will transform our input text into vectors of numbers. These numbers will represent each word that makes the training summary of the genre'.
Let's first create two dictionaries :

1) One to hold all words and the number of times they have been used across the corpus.
2) Another to hold all genres and the number of times they have been assigned a summary.

These two will be used later when we want to check what the model is learning about some of the most important words in the corpus.

Let us see the most common words and common genre's with their number of occurrences :



```
Common tags [('Fiction', 3857), ('Speculative fiction', 3468), ('Science Fiction', 2344), ('Novel', 2000), ('Fantasy', 1940)]
-----------------
common words [('one', 11621), ('time', 7573), ('also', 7335), ('two', 7134), ('new', 6891)]
```

The next step is to represent the words in the summary or description in a numerical way. There are different ways we can represent each word in a numerical way. **The most common are :**

1) Bag of words
2) TFIDF
3) Word Embeddings.

The later has recently attracted attention of many especially when you have a big corpus of millions of words(including n-grams). For this exercise, **I will use TFIDF** which is an improved version of bag of words which uses inversed logarithmic normalisation to penalise those words that are most frequent across different input text of our corpus.

Let's **use sklearn library to generate TFIDF vectors.** I will also use n-grams of size 1 and 2. you can experiment with size if you have enough RAM on your computer. Also keep in mind that increasing the size of n-grams without increasing the size of samples might generate a lot of features greater than the size of the sample. this can result into your model being high biased.

Steps to create the vectorizer are as follows :

1) Create TF-IDF vectorizer with a proper parameters choice
2) Fit the vectorizer on the train set
3) Transform the train, test, and validation sets and return the result

```
tfidf_vectorizer = TfidfVectorizer(min_df=5, max_df=0.4, ngram_range=(1, 2),token_pattern='(\S+)')
tfidf_vectorizer.fit(X_train)
X_train = tfidf_vectorizer.transform(X_train)
X_val = tfidf_vectorizer.transform(X_val)
X_test = tfidf_vectorizer.transform(X_test)
```

We have used the parameters with min_df of 5 which means that, when building the vocabulary ignore terms that have a document frequency strictly lower than given threshold of 5. Max_df of 0.4 is used to ignore terms that have a document frequency higher that 40%.

We are filtering out :

1) Rare words (occur less than in 5 titles)
2) Frequent words (occur more than in 40% of the titles).

We need to understand that most common words should be eliminated from the summary's. This is crucial since the performance our model will depend on the TFIDF representation of the data sets.
Let's check to see if the common words exist in the summary's

```
print('he' in tfidf_vocab)
print('back' in tfidf_vocab)
```

```
False
True
```

This tells us that the common words have been eliminated. We can move to training our classifier using the TFIDF vocabulary.

4. Training The Classifier

Now that our different data are ready. let's decide on the training technique.

Since this exercise requires to predict genres for each comment and a summary can have one or more than one genres, we will need to use a model that considers each input of tfidf vectors representing words of summary and its respective genre individually. Then it needs to be evaluated to find the probability that the input can be assigned the genre(output 1) or otherwise (output 0). we will use the "MultiLabelBinarizer" of sklearn to convert each tag into a binary form.
Example:
summary A - check genre 1 vs other genres --------------------------- output : other genre (0)
summary A - check genre 2 vs other genres --------------------------- output : other genre (0)
summary A - check genre 3 vs other genres --------------------------- output : genre 3 (1)
....
……

4.1 Build The Classifier

1) We will use "One Vs RestClassifier" to train each binary tag individually
2) We will use basic classifier called Logistic Regression as the upper layer.

It is one of the simplest methods, but often it performs good enough in text classification tasks. Traditional two-class and multi-class problems can both be cast into multi-label ones by restricting each instance to have only one label. On the other hand, the generality of multi-label problems inevitably makes it more difficult to learn.

An intuitive approach to solving multi-label problem is to decompose it into multiple independent binary classification problems (one per category). In an "one-to-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.

For Example: it is more like asking simple questions, say, "is the summary, Adventure genre or not", "is the summary, Fiction genre or not?", etc. Also there might be an extensive case of overfitting here, since most of the comments are unlabelled, i.e. most of the comments are clean comments.

```python
from sklearn.multiclass import OneVsRestClassifier
from sklearn.linear_model import LogisticRegression, RidgeClassifier

def train_classifier(X_train, y_train):
    """
      X_train, y_train — training data

      return: trained classifier
    """

    # Create and fit LogisticRegression wraped into OneVsRestClassifier.

    lr = LogisticRegression(penalty='l2') # use L2 to optimise
    ovr = OneVsRestClassifier(lr)
    ovr.fit(X_train, y_train)
    return ovr
```

4.2 Train and Predict the outcomes

There are two basic steps to using the classifier:

1) Training.
2) Classification or prediction.

Training is the process of taking content that is known to belong to specified classes and creating a classifier on the basis of that known content. Training is an iterative process whereby you build the best classifier possible,

```python
classifier_tfidf = train_classifier(X_train_tfidf, y_train)
```

Classification is the process of taking a classifier built with such a training content set and running it on unknown content to determine class membership for the unknown content. Classification is a one-time process designed to run on unknown content.

```python
y_train_predicted_labels_tfidf = classifier_tfidf.predict(X_train_tfidf)
```

Sample Output :

```
Title:  orrie finally going tie knot hes engaged marry jill hardy stewardess months orries also keeping company isabel kerr ex
showgirl orrie time available jill works international flights isabel time available longer performs rather occupies plush apa
rtment thats paid another gentleman friend visits two three times week isabel objects orries marriage plans taken personal pro
fessional belongings stashed apartment isabel threatens show jill thus quash marriage orrie asks archie get isabels apartment
find possessions get back archie enter apartment finds orries belongings isabels body archie withdraws meet orrie otherwise ke
eps news isabels sister stella later discovers body police find orries possessions apartment arrest suspicion murder meeting c
onsider whether orrie guilty wolfe archie fred unsure saul via convoluted reasoning concludes innocent wolfe undertakes demons
trate wolfe must determine knew isabels apartment orrie given archie names avery ballou pays bills stella fleming husband barr
y nightclub singer named julie jaquette archie visits stella barry learns stella frantic keep lid nature sisters living arrang
ements stellas concern isabels reputation tries claw archies face refers isabel doxy archie corrals reluctant ballou wolfe coe
rces cooperation threatening disclosure relationship isabel turns ballou already subjected blackmail someone named milton thal
es ballou thinks thales really orrie wolfe deduces thales true identity assumes isabels murderer wolfe sends saul bring julie
jaquette dances wolfes office miss jaquette puts performance first singing demanding see wolfes orchids displays cynicism rega
rding human behavior wolfe regards similar julie agrees act bait murderer nearly killed protection moved brownstone helps wolf
e archie force thales hand wolfe offers 50 000 cash assistance
True labels:    Detective fiction,Fiction,Mystery,Suspense
Predicted labels:       Detective fiction,Fiction,Mystery,Suspense
```

Not bad either. However our training data set is big and has a hundred of thousands of comments. the validation set is also big. we can ‘t evaluate each example by comparing labels. we need some numbers that can help us to understand how the model performed. we will use a combination of accuracy, f1 score and hamming score to evaluate the models.

5. Evaluation

To evaluate the results we will use several classification metrics:

1) F1 score
2) Hamming Score
3)

The evaluation measures for single-label are usually different than for multi-label. Here in single-label classification we use simple metrics such as precision and recall.

F1 Score:
The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

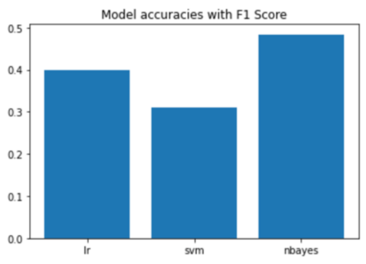
```
F1 = 2 * (precision * recall) / (precision + recall)
```

Let’s visualize the f-1 Scores:

```python
# Dictionary is made to plot the graph
f1_scores_dictionary = dict(zip(models, fscores))

print(f1_scores_dictionary)
```

```
{'lr': 0.3979673709548008, 'svm': 0.31085043988269795, 'nbayes': 0.483826247689464}
```



Conclusion :
We can see SVM has a better f1 loss.

Hamming Score:
Hamming-Loss is the fraction of labels that are incorrectly predicted, i.e., the fraction of the wrong labels to the total number of labels.

The normal accuracy like F1 score, Recall does not work appropriately because Recall, F1-Measure are designed for the binary class.

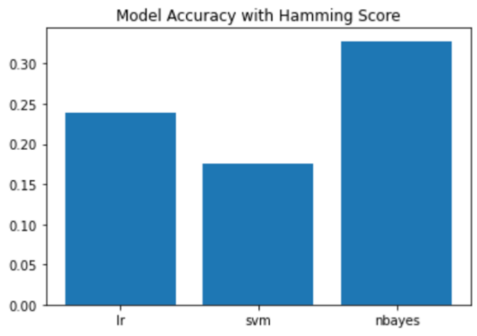
Lesser value of hamming loss indicates a better classifier. Hamming Score should be higher. Instead of Counting number of correctly classified data instance, Hamming Loss calculates loss generated in the bit string of class labels during prediction. It does XOR between general binary strings of class labels and predicted labels for data instance and calculates average across the dataset

```
# Dictionary is made to plot the graph
hamming_scores_dictionary = dict(zip(models, hamming_scores))

print(hamming_scores_dictionary)
```

```
{'lr': 0.23954111056187183, 'svm': 0.1761410446531554, 'nbayes': 0.32800811501071014}
```

Let's visualize the scores:

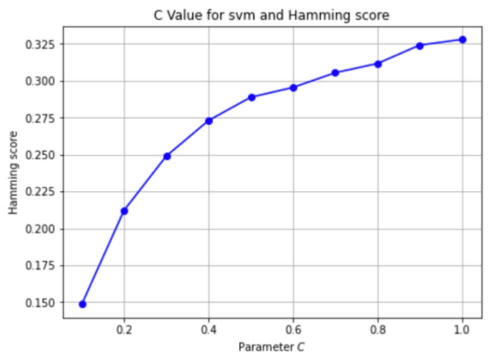


Conclusion:
We can see Naive bayes has a better hamming score. The higher the score, better it is.

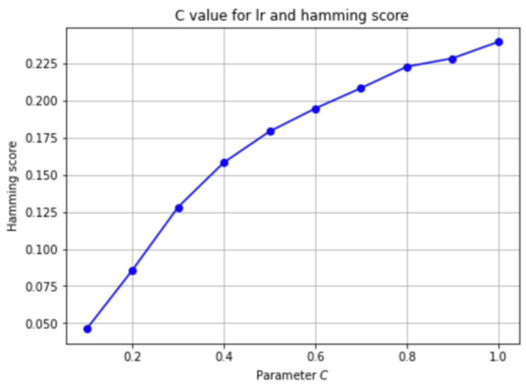
5.1 Hyper Parameter Tuning

Now, we will experiment a bit with training our classifiers by using weighted hamming-score as an evaluation metric. Moreover, we select to use the TF-IDF approach and try L1 and L2-regularization techniques in Logistic Regression with different coefficients (e.g. C equal to 0.1, 1, 10, 100).

SVM :



Conclusion:
It shows that the highest value is 0.325 at C= 1.0
Logistic regression:



Conclusion:
It shows that the highest value is 0.23 at C= 1.0.

6. Conclusion

We can conclude that for the provided dataset, Naïve Bayes gives the best results. Also if we want to use "logistic regression" instead of naive bayes then we can use the Hyper Parameter = 1. We can also observe that most of the genre are not being predicted. We can further use Word2Vec, Neural networks to improve the accuracy and predict the missed genres.

6.1 Best Model

Among all the 3 models the best hamming score is of Naïve Bayes with a score of 0.38. Hence, our final model should be trained using the Naïve Bayes classifier.