Solving Multi-label Classification Problem on Movie Genres

JONALIS BIN KAMIS, Ng Zhi Yun, Zhang Bo Lun

Abstract

This project is about exploring machine learning in using the provided data to predict the genre of a movie based on the plot. The data (movie_data.csv) comprises of 34,892 movie titles with a total of 9 columns ('Release Year', 'Title', 'Origin/Ethnicity', 'Director', 'Cast', 'Genre', 'Wiki Page', 'Plot', 'Number of words') Before feeding the data into the algorithm in predicting the genre, the datasets went through data treatment and preprocessing process to transform the data into a more cleaned and reliable dataset. Both processes include importing the relevant libraries, importing the dataset into the folder, managing missing or irrelevant datasets such as blanks, features extraction and splitting the datasets into training and testing datasets. To predict the final outcome, we have used the Scikit-Learn Chain Classifier as this can be applied to this multi-label classification problem. We have gone through a series of testing and parameter tuning and achieved a final F1 score of 0.5176.

1. Introduction

The goal of this project is to use machine learning methods to predict movie genres based on their plot descriptions. Each plot description will be tagged with one or more genres. Currently, assigning of genre names to particular movies or films are done after the entire movie has been completed. The genre to a movie is tagged manually through suggestions from readers through emails. This is too slow as people will want to watch movies based on specific genres that they personally want to watch immediately when the movies are released.

With advanced technology today, we are now able to make use of computer algorithms to help humans to automate this process. Supervised learning method is a method that we will be using in this project to help us with predicting the genre based on the plot description of a movie.

A movie can potentially have more than 1 genre tagged to it. The genre tagged to one movie can also be tagged to one or more other movies. Due to the nature of the datasets and the expected results, the team has chosen the multi-label classifier to perform the classification of the genres to the movies. The importance and usefulness of multi-label classification have been growing in the past few years due to the increase in its applications. In the past, binary classification or multi-class classification received much interest due to their simplicity to link classes to labels quickly. For binary classification, there are only 2 classes for example "Yes" or "No", "Positive" or "Negative". Multiclass classification is a process of classification with more than 2 classes and the assumption that each item is assigned to one and only one class. This is not applicable to our project as even though there are more than 2 classes, the classes can be assigned to one or more items in the dataset. For multi-label classification, tasks can be tagged with more than one class at the same time and the classes can still be tagged to more than one task throughout. Nowadays, multi-label classification has grown importance into other domains such as music categorization into emotion, semantic video annotation, direct marketing and semantic scene classification.

2. Exploratory Data Analysis (EDA) of genres and Plot

2.1 Overview of Data

We have downloaded our data from Kaggle. These data that the owner uploaded were scraped from the Wikipedia website. Most of the movie information is being updated on Wikipedia and is accurate as of now.

The raw dataset contains the following column description:

Table 1. Description of each column in the dataset

COLUMN	DESCRIPTION
RELEASE YEAR	Year in which the movie was released
TITLE	Movie title
ORIGIN/ETHNICITY	Origin of movie (i.e. American, Bollywood, Tamil, etc.)
DIRECTOR	Director(s) of the movie
MAIN ACTOR AND ACTRESSES	Main leads of a movie
GENRE	Movie Genre(s)
WIKI PAGE	URL of the Wikipedia page from which the plot Description was scraped

PLOT	Description of movie

There is a total of 9 columns and 34896 data point in the raw dataset. For the project we have only used the relevant data required. Hence, after data processing, we have a total of 28813 relevant data points. The cleaning process will be elaborated more in the later section of the report under pre-processing of data. Each of our data points will have the following attribute to be considered: Genre and Plot.

2.2 Genre

The maximum number of genres per data point is 25 and the minimum is 1. On average, there are approximately 1 (~1.38) genre per movie. Figure 1 below shows the summary of number of genres for each movie. As we can see from the figure, there are more than 20000 movies have only 1 genre tagged to it (actual number is 20172). There is also a total number of 7183 movies with two genres and 964 movies with 3 genres

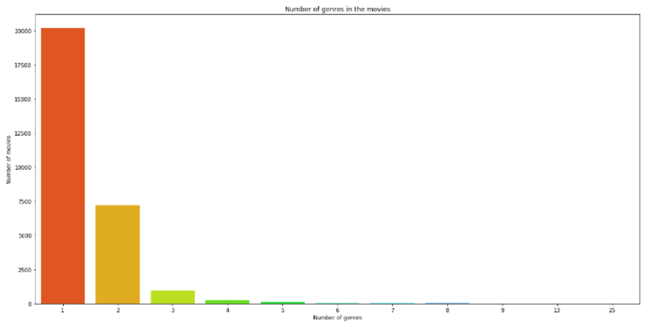


Figure 1. Number of genres for each movie

There is a total of 662 unique genres. Figure 2 below shows the top 20 genres.

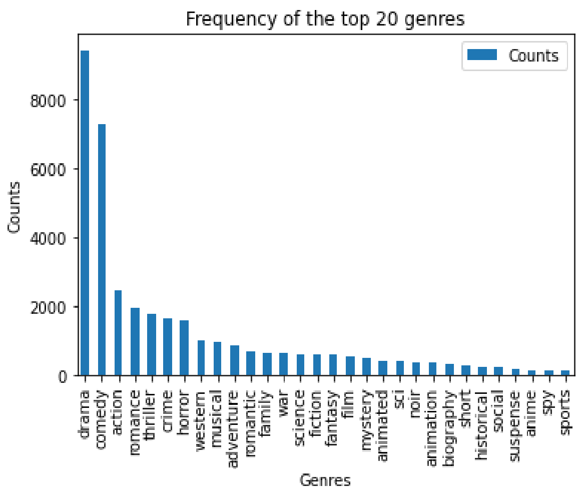


Figure 2. Frequency of the top 20 genres

The top 3 genres are ‘drama’, ‘comedy’, ‘action’ and the number of times it appears are 9416, 7276 and 2447 respectively. On average, the number of counts that each genre appear is 60 times. There are some least popular genre that only appear 1 time throughout the dataset. Some of the examples are ‘madhu’, ‘kiran’ and ‘kota’.

2.1 Plot

It is represented by a string to describe the content of a movie. The number of words on average for a plot is 383. The maximum number of words of a plot is 5224 while the minimum is 1. Figure 3 below shows the top 10 words found in the plot. The top 3 most frequent words are ‘find’, ‘him’ and ’one’ with a count of 25503, 24822 and 23812 respectively. There are also some least common words which only appear once in the whole dataset. Some of the examples are ‘inslee’ , ‘musser’ and ‘weathervane’.

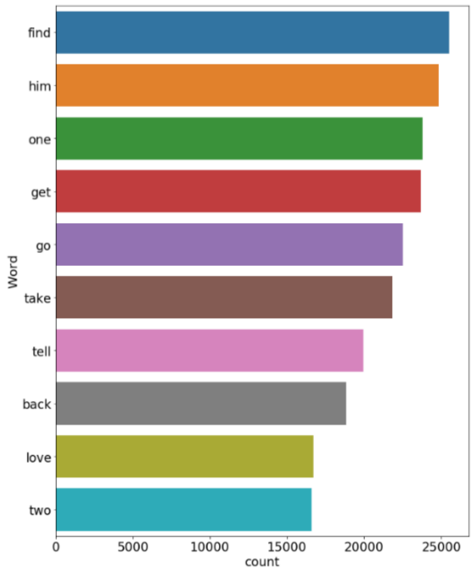


Figure 3. Frequency of the top 10 words in plot

3. Preprocessing of data

We would need to pre-process the data so that only meaningful features will be extracted.

3.1 Preprocessing on Genres

After observing the genre column, we have noticed that there are many non-standard genres and many of such genres only appear a few times in the whole dataset. Numbers and special characters are also present in the genre column, such data should not appear in the data set as they will affect the prediction result significantly. Genres with less than 3 characters are removed as well since the words are not actual genre names which will affect prediction.

These are the steps that have been done in cleaning genres data:

1. Remove Special Characters

2. Remove numbers
3. Remove words with less than 3 characters
4. Remove NULL and Genre name equals to unknown
5. Tokenization, this was done as part of Vectorization process

Besides above-mentioned methods, we have also created a list of genres that appear less than 100 times in the whole data set. The whole dataset is looped through the list and any genre appears in the list was removed from the genre list. At the end of the cleaning process on the movie genre column, we have a total number of unique 34 genres.

3.2 Preprocessing on Plot

In the plot we notice that not all the words are important to determine the genre of the movie. Moreover, the text are not standardized in various format.

These are the following steps that have been done to the test:

1. Change all to Lowercase
2. Remove English Stop word
3. Lemmatize the words
4. Remove punctuation

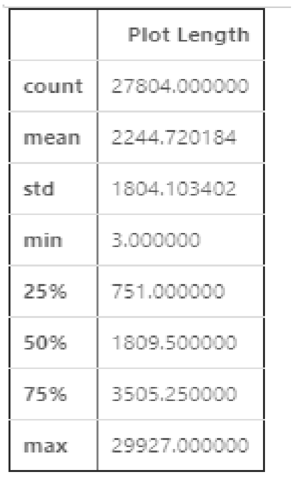
Sample plot before cleaning: 'The film is about a family who move to the suburbs, hoping for a quiet life. Things start to go wrong, and the wife gets violent and starts throwing crockery, leading to her arrest.'

Sample plot after cleaning: 'film family move suburbs hoping quiet life thing start go wrong wife get violent start throwing crockery leading arrest'

After processing the plot through the steps above, the length of the plot would be expected to decrease. The table below shows the decrease in the length of the plot. Before the text processing there was a average of 2244 characters and after processing there is only 1514 characters left in the plot.

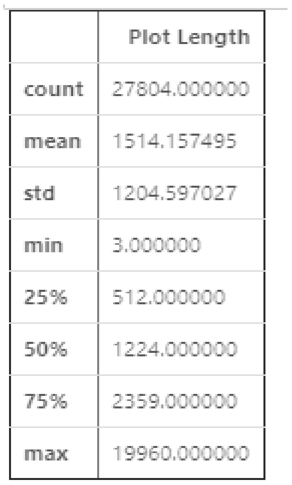
Table 2. Distribution of length of plot summary length

Before Cleaning



	Plot Length
count	27804.000000
mean	2244.720184
std	1804.103402
min	3.000000
25%	751.000000
50%	1809.500000
75%	3505.250000
max	29927.000000

After Cleaning



	Plot Length
count	27804.000000
mean	1514.157495
std	1204.597027
min	3.000000
25%	512.000000
50%	1224.000000
75%	2359.000000
max	19960.000000

4. Feature Engineering

4.1 Genre

Since our classification labels consist of english words instead of numeric values, feature extraction is required for the labels as well. Bag of words were used for the feature extraction for genre. The genre string is first being tokenized into individual words. Secondly, it will count the occurrence of the token in each of the data points. Finally, normalizing and weighing the importance of the tokens based on the occurrence. The relative position of the genre information will be ignored. This was implemented with python library function CountVectoriser() from sklearn library.

Count vectorization is also known as one hot encoding. It creates vectors which have the same dimensionality with the size of our vocabulary. If the genre data features that token, we put a one in that dimension. Each time the word appears again, we increase the count, leaving 0s everywhere else that the word does not appear. A matrix of 0s and 1s are formed for each genre at the end of vectorization. (Heidenreich, 2018)

4.2 Plot

4.2.1 TFIDF

Term Frequency Inverse Document Frequency (TF-IDF) will be used for the feature extraction for plot. Term Frequency(TF) measures how common a word appears in a particular plot summary by the frequency that it appears. Higher TF value means the word appears more frequently in a plot. IDF measures how unique a word is in the plot by how infrequent the word appears across all plot summaries. If a word appears frequently in all plots, it will have a small IDF value and means it is not a very meaningful word across the set of plots. Hence, TF-IDF score is calculated by the product of TF and IDF. A high score will indicate that the word occurs frequently and provide the most information in that specific plot. (Bedi, 2018) Sklearn library TfidfVectorizer() will be used for the implementation. Vectorizer is applied to both training dataset and test dataset. In the project, we have tuned two parameters of the method: max_features and ngram_range. Max_fetures define the maximum number of features used in the vectorization process. Parameter ngarm_range specifies the lower and upper boundary of the range of n-values for different n-grams to be extracted. (1,1) means a unigram, (1,2) means both unigram and bigram. We have tried both char and word analyzer on the n-grams parameter. A detailed tuning process will be discussed in section 6.2.

4.2.2 WORD EMBEDDING

Word2Vec is another technique in which we can convert a word to numerical vectors. In the project, we have used a pre-trained W2V model by Google to convert words to its corresponding vector form. For a given plot summary, we have converted each word to vectors, sum vectors up and find the mean value to represent an average word2vec word embedding for a given plot summary.

Assume that we have n words in a plot summary {w1,w2,w3,w4,w5,w6 … , wn}. We will convert each word to a vector, add the vectors together and divide by the total number of words (n) appearing in that particular plot. Hence, our final vector will look like [word2vec(w1) + word2vec(w2) + word2vec(w3) …. + word2vec(wn)]/n. (Paul, 2019) Code snippet 1 shows the pretrained Google model which we have used how it is applied as our vectorization model. Since the vectorization and calculation process took an extremely long time (total of 10 hours for training and testing set), we have saved the vectorized vectors in two files. We can load the data any time during training and testing process.

Code snippet 1 Word2Vec model

```
from gensim.models import Word2Vec
from gensim.models import KeyedVectors


word2vec_model=KeyedVectors.load_word2vec_format('
GoogleNews-vectors-negative300.bin', binary=True)
word2vec_words = list(word2vec_model.wv.vocab)
```

5. Machine Learning Models

5.1 Dealing with multi-label

Multi-label problem has a slightly different strategy in training and testing. One extra step is involved in resolving the multi-labels. Models are implemented to treat each label separately and combine the result during the training process or at the end of training process. Three models are explored in this project. Details are explained in below sections.

5.1.1 ONEVSREST

OneVsRest breaks down a multi-label problem into multiple binary classification problems. There will be N classifiers for N labels. In testing, the model classifies the input with the highest score among all classifiers. For a given movie during training and testing, the model will check whether this plot is drama? Whether it is comedy? Or whether it is action. It breaks down a multi-label problem into three binary classification problems. At the end of the training process, it combines the result of these three problems into one single output. Hence, it is about fitting one classifier for each class. The class is fitted against all the other classes for every classifier.

5.1.2 BINARYRELEVANCE

Binary Relevance is very similar to OneVsRest model. It transforms a multi-label classification problem with n labels into n single label separate classification problems. Every individual classifier will predict whether a genre is true or not. Hence, each classifier acts as a binary classifier. At the end of the training process, the result of all the binary classifiers are combined together to form a multi-label output.

5.1.3 CLASSIFIERCHAIN

Classifier Chain is another common way in resolving multi-label problems. It considers one label at a time like binary relevance but it also takes label dependency into consideration. The output of each classifier is fed into the input of the subsequent classifier. For a dataset with n labels, it has a total number of n classifiers, Cn-1 is fed into input of classifier Cn. Hence, each label in a dataset is not trained individually.

5.2 Training Models

5.2.1 SGDCLASSIFICATION

SGDClassification is a classifier by sklearn. SGD stands for stochastic gradient descent. SGD is actually an optimization method which improves the speed of training models. When we apply parameter 'loss' as log, SGD classifier will use Linear Regression as the training model. It is better than linear regression as it has a faster speed in processing large dataset. Logistic Regression model predicts the possibility of a class by utilizing a logit function. Logit function is a log function of odds in favour of an event. (SRIVASTAVA, 2015) SGD classifier has other loss functions as well, in the project, we have mainly focused on the Logistic Regression model.

5.2.2 NAÏVE BAYES

Naïve Bayes is a simple classification method based on Bayes theorem. It has a naïve assumption that the probability of a word occurrence is independent of each other. To classify a set of words into a category, it uses the product of the probability of each individual word to predict the likelihood of a given sentence. (Chávez, 2019)

It is a simple algorithm that works well on small datasets. However, it has the disadvantage as it assumes each feature is independent of other features. This is very unlikely in processing natural languages. (Sawla, 2018)

5.2.3 XGBoost

XGBoosting stands for extreme gradient boosting. Boosting model is a type of supervised learning ensemble model which can convert weak learners to strong learners by combining several learners together. (BANSAL, 2018) XGBoosting is a more efficient implementation of gradient boosting with improved performance and speed. Although it has a relatively faster speed with competitive accuracy, it is subjective to overfitting if the parameters are tuned properly. Given the number of parameters which XGBoosting has, it would take a long time in performance tuning. (Gupta, 2020)

5.2.4 K-nearest neighbour (KNN)

KNN is a lazy learning algorithm which means no training data is required for model generation. We simply define a value of K which is the number of nearest point to a testing data. We then calculate the distance between new point and surrounding points to find K nearest data. Within K points, the new data point belongs to the group with most points. (Navlani, 2018) In this project, we have implemented ML-KNN as an adapted algorithm which adapts a single label problem into multi-label algorithm by changing cost function. skmultilearn.adapt library is used for MLkNN.

6. Model Evaluation

6.1 Evaluation methods

With all the training models, we need to have ways to evaluate the accuracy of our models. There are several ways chosen and Micro averaged F1 score is selected as the main evaluation method due to its compatibility with imbalanced labels.

6.1.1 Accuracy

Accuracy measures all the cases that are classified correctly. It is used when all the cases are equally important. (Huilgol, 2019)

$$Accuracy = \frac{True\ Positive\ +\ False\ Positive}{True\ Positive + False\ Positive + True\ Negative + False\ Negative}$$

Formula 1 Accuracy

6.1.2 Precision

Precision measures the true positive cases over all the predicted positive cases. This is useful when the cost of false positive is high. (Huilgol, 2019)

$$Precision\ = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Formula 2 Precision

6.1.3 Recall

Recall measures the true positive cases against all the actual positive cases. This is used when the cost of false negative is high. (Huilgol, 2019)

$$Recall\ = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

Formula 3 Precision

6.1.4 Hamming Loss

Hamming loss basically measures the accuracy in a multi-label classification task. It is the fraction of wrongly predicted class labels over total number of labels. When all the tags are classified correctly, Hamming Loss will be 0. The formula for Hamming loss is given by:

$$HammingLoss(x_i, y_i)\ = \frac{1}{|D|}\sum_{i=1}^{|D|}\frac{xor(x_i, y_i)}{|L|}$$

Formula 4 Hamming loss

In the above formula,

|D| is the total number of test data.

|L| is the total number of unique genres we have.

Yi is the ground truth.

Xi is the predicted values for each data point.

To explain hamming loss in a multi-label classification, we will have an example of three genres g1, g2 and g3. Yi is the ground truth for the data point i and it's represented by the binary vector [1,1,1], where the data point i belongs to all the three tags simultaneously. Xi is the predicted binary vector [1,0,1] for the data point i given by the model. It means the model has made correct predictions on genres g1 and g3 but it failed to predict on genre g2. Hence, the XOR of [1,1,1] and [1,0,1] is [0,1,0]. Corresponding Hamming Loss is 1/3 since we have a total number of 3 genres. (Paul, 2019)

6.1.5 F1-Score

Micro averaged F1 score is the main evaluation method which we have used in this project, it is a harmonic mean between the micro averaged recall and micro averaged precision. The formulae for micro averaged precision score is given by:

$$F1_{micro} = \frac{2Precision_{micro}Recall_{micro}}{Precision_{micro} + Recall_{micro}}$$

Formula 5 F1 Score

In case of micro average F1 score, there is a weightage on the frequency of a label. We still have an example of three genres g1, g2 and g3 spread across the entire dataset. We assume 90% of our movies data has genre g1, 1% of the data tagged with g2 and 80% of the data has g3 as well.

Hence, true positive value and false positive value of g2 is small since it only has a small proportion. However, the contribution of g1 and g3 is much higher as they are more frequent in the dataset. (Paul, 2019)

To conclude, micro averaged f1 score considers label frequency as well. Although the weighted recall and precision score of genre g2 is extremely low and we have a high precision and recall of genres g1 and g3, we still can achieve pretty high micro averaged F1 score due to the higher frequency of genres g1 and g3. Since our genres are very imbalanced as explained in the data analysis section, we have decided to use micro averaged F1 score works as our main evaluation method.

6.2 Parameters tuning and Data Evaluation

In this section, we will perform various rounds of model comparison and parameter tuning on different methods used. In the code development, we have used precision, hamming loss, accuracy and F1 score as our evaluation method. Due to the space limit in report, we have only included result with F1 score and accuracy.

6.2.1 RESULT ON DIFFERENT TRAINING MODEL

We have tested on four different models on the movie data. All four models are tested with CountVectorizer for genre and TFIDF for movie plot. Max_features of TFIDF was kept at 10000 with char n_grams and ngram_range of (6,6). Multilabel handling method used was OneVsAll. As seen in table 3 below, XGBoost has achieved a higher accuracy value of 0.3625 but its F1 score is very low as compared to others. SGDClassifier on the other hand, has a much higher F1 score of 0.3429. Since we have decided to use F1 score as our main evaluation method, SGDClassifier was used as the model for movie genre classification problem.

Table 3. Result on various model testing

MODEL	MICRO F1 SCORE	ACCURACY
SGDCLASSIFIER	0.3429	0.273871605826
NAÏVE BAYES	0.2996	0.32997662290
XGBOOST	0.1724	0.362524725768
KNN	0.2182	0.321704729365

6.2.2 RESULT ON DIFFERENT MULTILABEL HANDLING METHOD

In this section, we have explored different multilabel handling methods. Parameters of TFIDF and Count Vectorizer were kept the same as section 6.2.1. Model used for testing was SGDClassifier. As observed from table 4, Binary Relevance had the highest accuracy value with 0.3823 but lowest F1 score of 0.2917. Classifier Chain had the highest F1 score of 0.3838 but relatively lower accuracy of 0.2990. As a result, we can conclude to use Classifier Chain as the better method in handling multilabel problems.

Table 4. Result on various multilabel handling method

MODEL	F1 SCORE	ACCURACY
ONEVSALL	0.3429	0.273871605826
BINARY RELEVANCE	0.2917	0.382305340766
CLASSIFIER CHAIN	0.3838	0.299046934004

6.2.3 PARAMETER TUNING ON TFIDF VECTORIZER

In this section, we have gone through a series of tuning on TFIDF vectorizer. The main tuning was on max_features and ngram_range parameters. We have also explored both char analyzer and word analyzer. The models used were SGDClassifier and Classifier Chain.

The first parameter we have looked into was the analyzer. We have tried both word and char. For the word analyzer, F1 score was 0.3925 with n_gram (1,3). Since it has a lower value than that of char n_gram, we have decided to continue with char n_gram. Regarding the max_features, we have tested a range from 5000 to 20000, the best F1 Score was achieved at 0.4052 with a max_feature of 15000. This means first 15000 features will be used in model prediction. We have also tested various n_gram options. Upper limit was set at 6 first and lower limit was increased from 1 to 6. At the end of testing, we have noticed that the best performance occurred when ngram_range was (3,6) with a F1 score of 0.4210.

Table 5. Result on TFIDF parameter tuning 1

MAX_FEATURE	F1 SCORE	NGRAM RANGE	F1 SCORE
5000	0.3758	1,6	0.3026
10000	0.3753	2,6	0.4045
15000	0.4052	3,6	0.4210
16000	0.4034	4,6	0.3729
18000	0.3337	5,6	0.3350
20000	0.3761	6,6	0.4065

With the lower bound fixed to 3, we have explored various upper bound again. The result is shown in table 6 below. Best upper value as achieved at 8 where the F1 score was 0.4509. Until now, we have improved our F1 score from 0.3429 without any tuning to 0.4509.

Table 6. Result on TFIDF parameter tuning 2

NGRAM RANGE	F1 SCORE	NGRAM RANGE	F1 SCORE
3,7	0.4215	4,8	0.4133
3,8	0.4509	4,7	0.4306
3,9	0.4142	4,6	0.3740
3,10	0.3968	4,5	0.3714

6.2.4 PARAMETER TUNING ON COUNTVECTORIZER

With the tuning of TFIDF vectorizer parameters, we then went ahead to tune on Count vectorizer parameters. We have mainly focused on the ngram_range value. As seen in the table below, the highest F1 score was obtained when F1 ngram_range was 3. Hence, we will use n_gram size equals to 3 as the value.

Table 7. Result on Count vectorizer parameter tuning

NGRAM_RANGE	F1 SCORE	ACCURACY
3	0.4509	0.35533177486063655
4	0.3796	0.2855601510519691
5	0.4145	0.31541089732062577
6	0.3606	0.26002517532817837

6.2.5 PARAMETER TUNING ON TRAINING MODEL

As the last step, we were going to tune on parameters in SGDClassifier. The two main classifier we have focused on were loss function and alpha value which is a constant that multiplies the regularization term. We have examined six different loss functions and the algorithm with best performance was ‘log’ which is a logistic regression model.

We have also tested on Alpha value of the model. In the first round of testing. We found that 0.0001 achieved the best result with F1 score of 0.4520. In the subsequent testing, we have worked on the alpha value with a range of 0.0001 to 0.0009. The highest F1 score value was achieved at Alpha = 0.0002. Hence, we have concluded that 0.0002 would be used as the final alpha value.

Table 8. Result on SGDClassifier parameter tuning 1

LOSS	F1 SCORE	ALPHA	F1 SCORE
HINGE	0.4018	1	0.4377
LOG	0.4509	0.1	0.3036
MODIFIED_HUBER	0.4386	0.01	0.3114
SQUARED_HINGE	0.3940	0.001	0.4491
PERCEPTRON	0.2659	0.0001	0.4520
SQUARED_LOSS	0.3219	0.00001	0.3506

Table 9. Result on SGDClassifier parameter tuning 2

NGRAM_RANGE	F1 SCORE
0.0002	0.4643
0.0003	0.4400
0.0004	0.4119
0.0005	0.4509
0.0006	0.4313
0.0007	0.4465
0.0008	0.4505
0.0009	0.4133

6.2.6 EXPERIMENTING ON WORD EMBEDDING

At the last step, we have experimented on word embedding instead of bag of words embedding with TFIDF vectorizer. As mentioned in section 4.2.2, we have added vector of each word in a sentence together and divided by the total number of words in a plot. The result was saved in pickle file. This was because each vectorization process consumes a great amount of time (approximately 8 hours on training set and 2 hours on test set). We cannot afford to perform the vectorization process every time. We simply load the data from files when we need them. By using Word2Vec vectorizer, our performance on F1 score has improved significantly with a result of 0.5176 as compared to 0.4643 when we used TFIDF vectorizer.

At this point, we have tested all the models and their corresponding parameters. The final F1 score we have achieved was 0.5176. Below is the list of models and the parameters used.

Table 10. Result of parameter tuning

MODEL	PARAMETERS	VALUES
COUNT VECTORIZER	max_features	3
WORD2VEC	NA	NA
SGDCLASSIFIER	alpha	0.0002
	loss	log
CHAIN CLASSIFIER	N.A	N.A

7. Conclusion

At the end of the project, we have completed the multi-label classification problem in movie genres prediction. We have applied various techniques through the whole project such as exploratory data analysis, natural language processing, feature extraction and machine learning. We also had learnt the difference in handling multi-label classification as compared in binary classification problem during Kaggle competition. Extra steps are required to convert data into a form that can be fit into various models. There are also models like ML-KNN which does not require special handling. Word embedding is great tool in terms of vectorizing our input text data but maintaining the dependency between words as well. It helps to improve the F1 score significantly and helped us to achieve a final score of 0.5176. Although we have explored various training models, we have not experimented anything related to neural networks. This can be an area which we can look into in the future.

References

BANSAL, S. (2018, April 23). *A Comprehensive Guide to Understand and Implement Text Classification in Python*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2018/04/a-comprehensive-guide-to-understand-and-implement-text-classification-in-python/

Bedi, G. (2018, November 9). *A guide to Text Classification(NLP) using SVM and Naive Bayes with Python*. Retrieved from medium: https://medium.com/@bedigunjit/simple-guide-to-text-classification-nlp-using-svm-and-naive-bayes-with-python-421db3a72d34

Chávez, G. (2019, Feburary 28). *Implementing a Naive Bayes classifier for text categorization in five steps*. Retrieved from TowardsDataScience: https://towardsdatascience.com/implementing-a-naive-bayes-classifier-for-text-categorization-in-five-steps-f9192cdd54c3

Gupta, S. (2020, Feburary 2020). *Pros and cons of various Machine Learning algorithms used in Classification*. Retrieved from TowardsDataScience: https://towardsdatascience.com/pros-and-cons-of-various-classification-ml-algorithms-3b5bfb3c87d6

Heidenreich, H. (2018, August 24). *Natural Language Processing: Count Vectorization with scikit-learn*. Retrieved from towards data science: https://towardsdatascience.com/natural-language-processing-count-vectorization-with-scikit-learn-e7804269bb5e

Huilgol, P. (2019, August 24). *Accuracy vs. F1-Score*. Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/accuracy-vs-f1-score-6258237beca2

Paul, S. (2019, June 16). *A detailed case study on Multi-Label Classification with Machine Learning algorithms and predicting movie tags based on plot summaries!* Retrieved from medium: https://medium.com/@saugata.paul1010/a-detailed-case-study-on-multi-label-classification-with-machine-learning-algorithms-and-72031742c9aa

Sawla, S. (2018, June 9). *Introduction to Naive Bayes for Classification*. Retrieved from Medium: https://medium.com/@srishtisawla/introduction-to-naive-bayes-for-classification-baefefb43a2d

SRIVASTAVA, T. (2015, October 4). *Building a Logistic Regression model from scratch*. Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2015/10/basics-logistic-regression/

Kartik Nooney, (2018, June 8), *Deep Dive into Multi-Label Classification ..! (With detailed Case Study)*. Retrieved from Medium: https://towardsdatascience.com/journey-to-the-center-of-multi-label-classification-384c40229bff

Navlani, A. (3 August, 2018). *KNN Classification using Scikit-learn*. Retrieved from DataCamp: https://www.datacamp.com/community/tutorials/k-nearest-neighbor-classification-scikit-learn

Github repository link is:
https://github.com/zbl09/H6751-Group-Assignment---01