**CSE-587 Assignment 3**

**Predicting Movie Genres Based on Plot Summaries**

**Ankita Das (50317491)**

**Rohit Lalchand Vishwakarma (50320038)**

**Manisha Biswas (50317483)**

**Abstract**

This project depicts the development and simulation of predicting movie genres based on plot summaries by using Apache Spark. We have been given 3 datasets, train, test and mapping. The task of predicting the genre is essentially a multi-label classification problem. A movie can have multiple genres associated with it. So, our model should be able to predict all the genres associated with the movie.

Video Link - <https://youtu.be/YC_itTVl3ME>

* **System and Software Dependencies:**

We are using the CSE 587 VM provided to us. The Software dependencies for the VM are as follows:

* Install Pandas if it not installed by default
* JDK 8 (As the VM has JDK 11 installed by default)

1. Check the installed Java version

For this, run the following in the terminal:

|  |
| --- |
| java -version |

By default the VM has JAVA 11 and we need to downgrade it to JAVA 8

1. Install JDK 8

For this, run the following in the terminal:

|  |
| --- |
| sudo apt install openjdk-8-jdk |

1. Set the $JAVA\_HOME environment variable

For this, run the following in the terminal:

|  |
| --- |
| sudo vi ~/.bash\_profile |

It will open the file in vi editor. Then, in a new line type the new JAVA\_HOME Path as below

|  |
| --- |
| JAVA\_HOME="/usr/lib/jvm/java-8-openjdk-amd64" |

Type wq! and exit. This will save the edit in the file. Later, in the terminal print the JAVA\_HOME

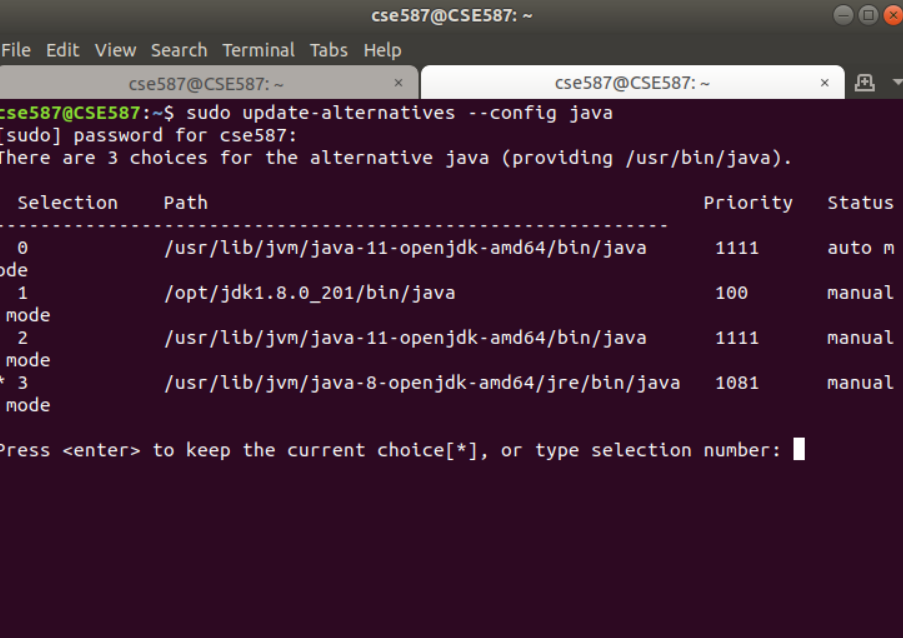
|  |
| --- |
| echo $JAVA\_HOME |

1. Set the JDK-8 as the Java Version

For this, run the following command in the terminal

|  |
| --- |
| !sudo update-alternatives --config java |

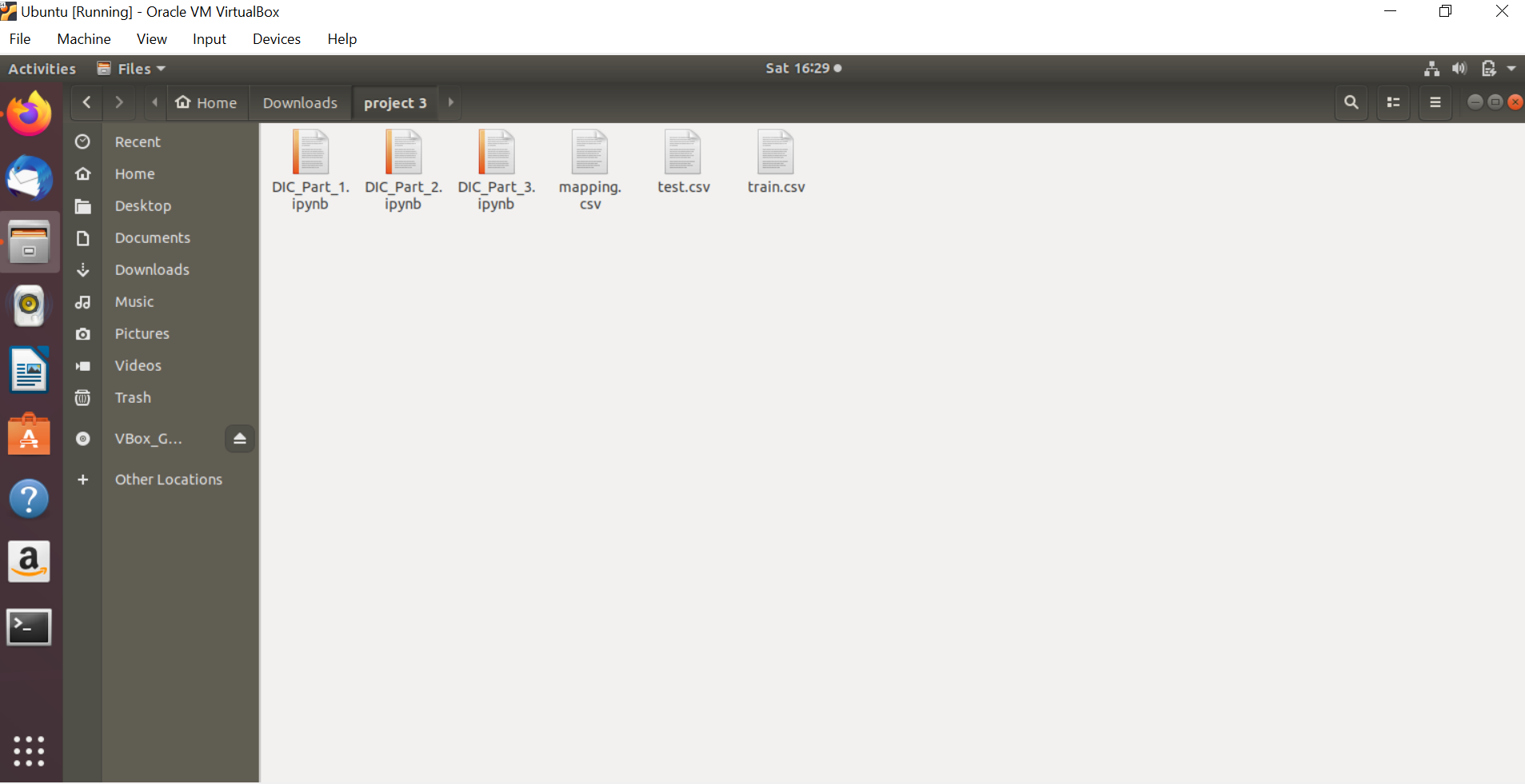
This should show the list of Java version installed in the VM as below:



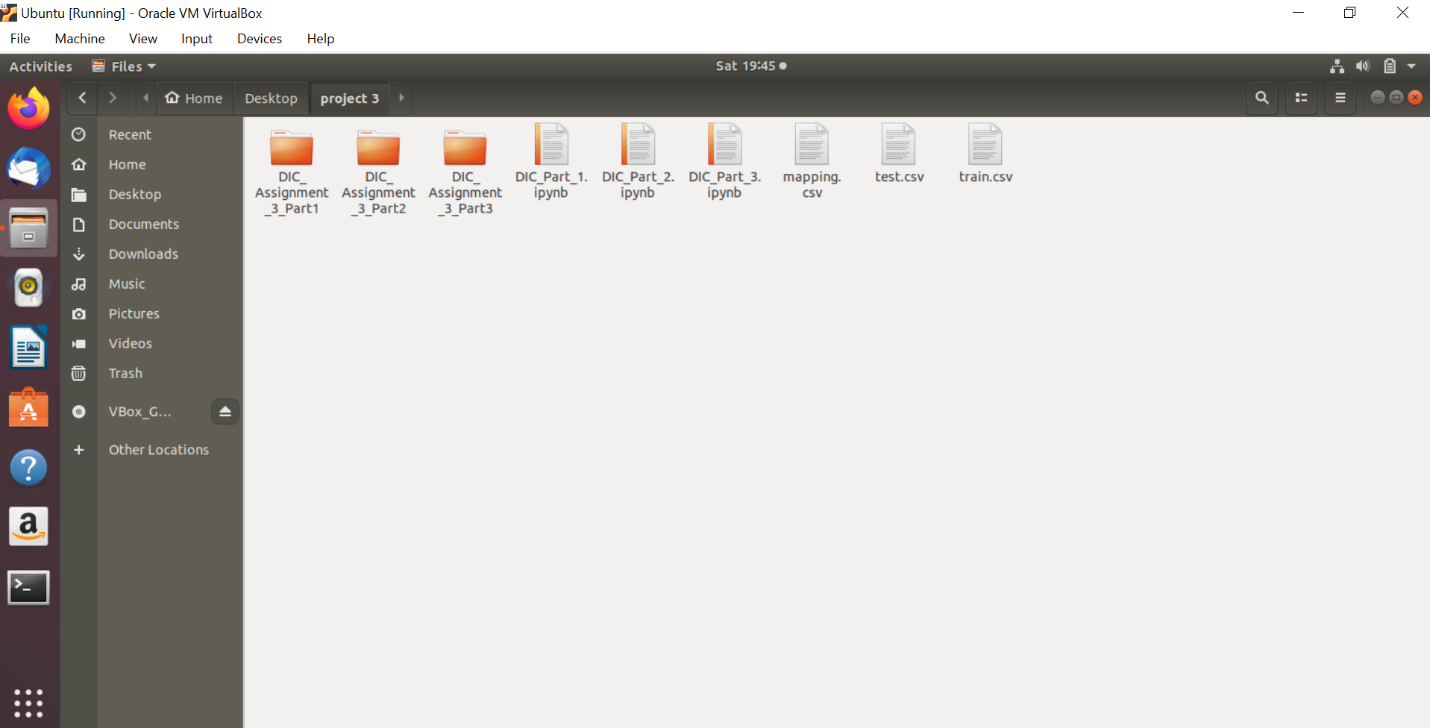
Select the Java 8 version and you are all set!

* **Project Configuration:**

1. Create a folder project 3 and place the train.csv, test.csv and mapping.csv in the same folder. Also put all the three DIC\_Part1.ipynb, DIC\_Part2.ipynb, DIC\_Part3.ipynb files in the same folder

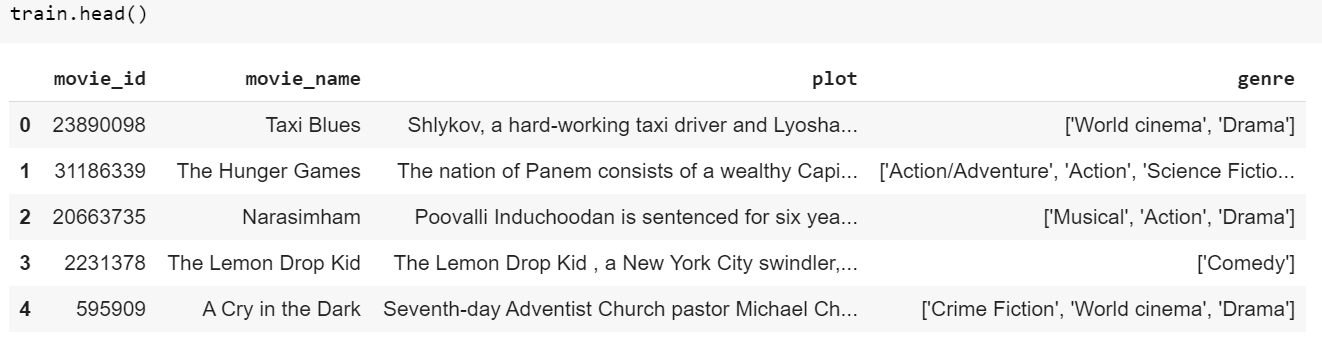


Run the files that will save three prediction csv in the same folder.

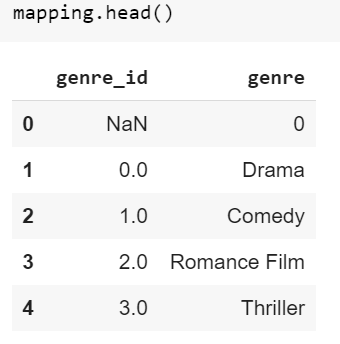


* **Dataset Definition**

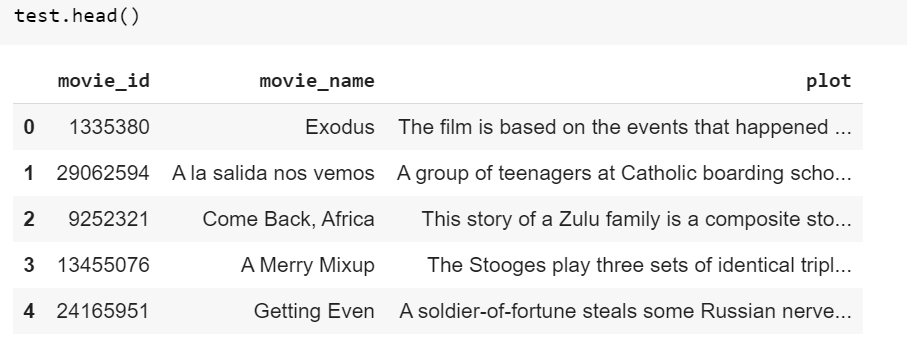
The train dataset has been provided with 31108 records with plot summaries. The dataset is consisting of 4 columns, namely movie\_id, movie\_name, plot, genre.



The mapping of the genre to the string index is given by the mapping dataset as below:



We need to predict the genre for the movie plots given in the test dataset as below:



As we are doing this project in pyspark, so converted all of the above-mentioned datasets into pyspark dataframe.

* **Data Preprocessing:**

After all the data were collected, following preprocessing was done on it using ‘regexp\_replace’ from pyspark library.

* Any/all use of http and HTML tags were removed.
* Punctuation, Numbers and special characters were removed.
* **Machine Learning pipeline**

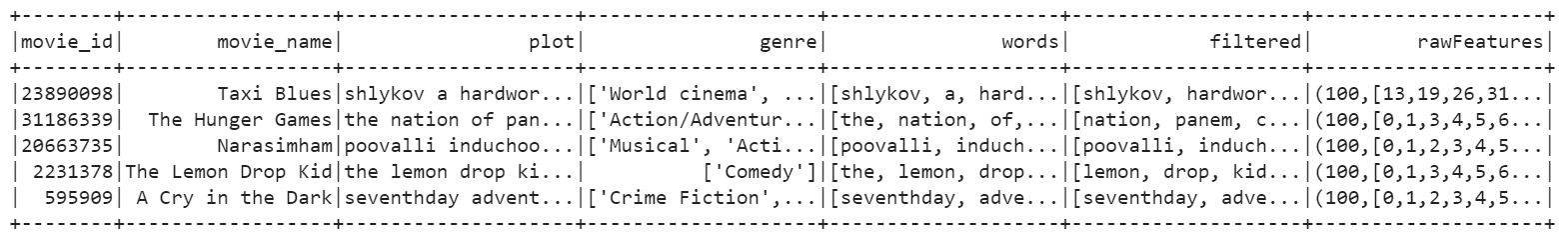
Tokenization was done on cleaned data using Tokenizer class to get individual terms from the plots and all the stop words which were irrelevant to the context were removed using ‘StopWordsRemover’ from pyspark.ml.feature library.

### **Text to Features Conversions:**

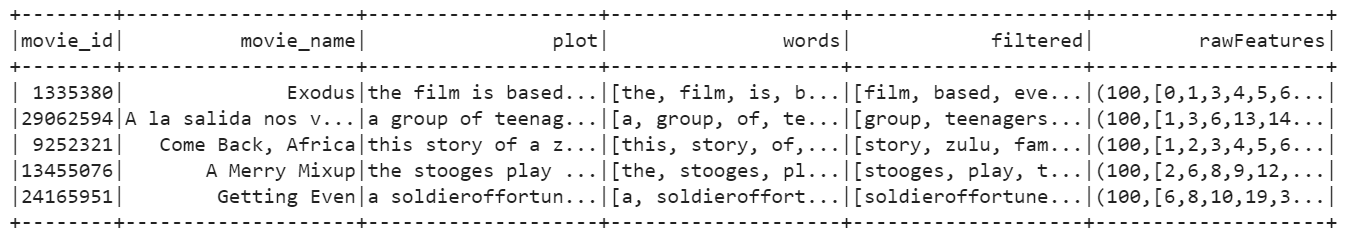
We need to extract features from the cleaned version of the movie plots data. For this purpose we have used 3 different approaches in 3 parts of the assignment.

1. **Part 1: Basic Model**

For the 1st part, we need to create a term-document matrix from the plots and use these as feature vectors for the machine learning model. We have used HashingTF, which takes sets of terms and converts those sets into fixed-length feature vectors. The terms are mapped to indices using a Hash Function. The hash function used is MurmurHash 3. The term frequencies are computed with respect to the mapped indices. We have used the 100 most frequent words in the data as our features. Same technique has been used for train and test data to extract the features.

The prepared **feature vectors for train** is as below: 

The prepared **feature vectors for test** is as below:

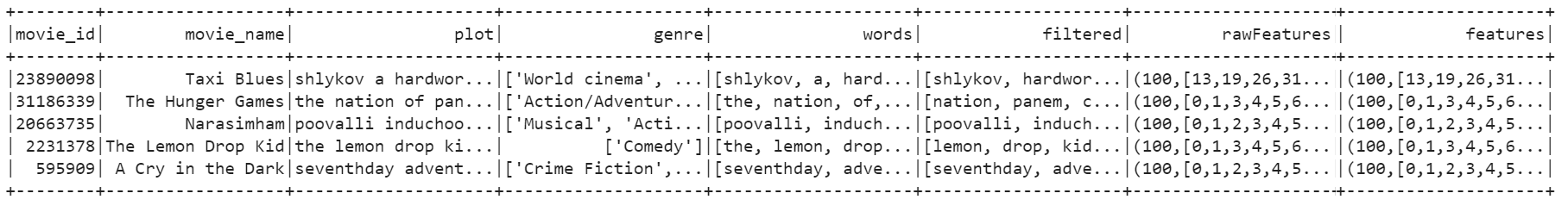


1. **Part 2: Using TF-IDF**

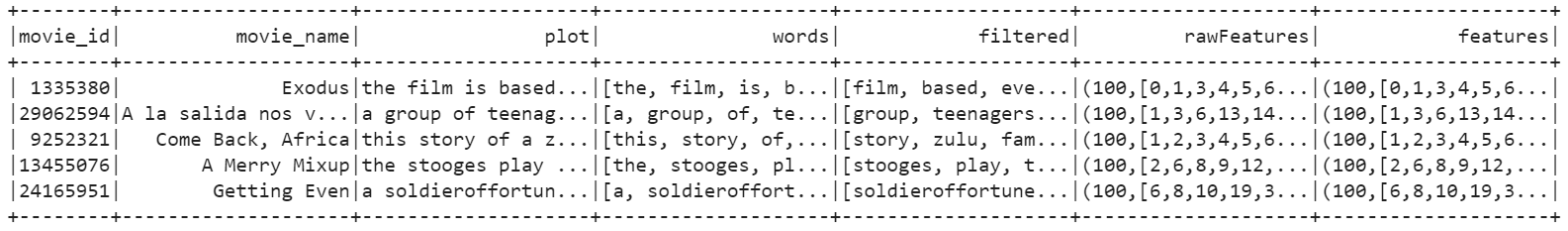
For the 2nd part, we implemented Term Frequency-Inverse Document Frequency (TF-IDF) based feature engineering technique. TF picks the most frequently occurring terms. We also want a measure of how unique a word is i.e. how infrequently the word occurs across all documents (inverse document frequency or idf). So, the product of tf & idf (TF-IDF) of a word gives a product of how frequent this word is in the document multiplied by how unique the word is with respect to the entire corpus of documents. Words in the document with a high tf-idf score occur frequently in the document and provide the most information about that specific document.

We have used **HashingTF-IDF** and used the 100 most frequent words in the data as our features. Same technique has been used for train and test data to extract the features.

The prepared **feature vectors for train** is as below:



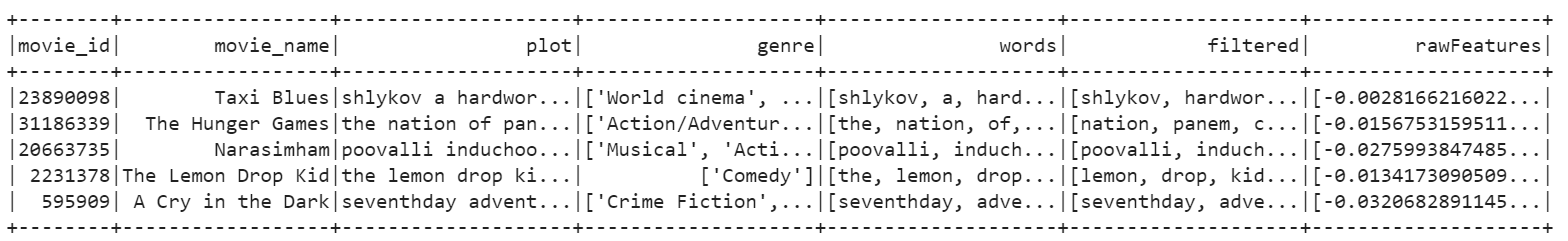
The prepared **feature vectors for test** is as below:



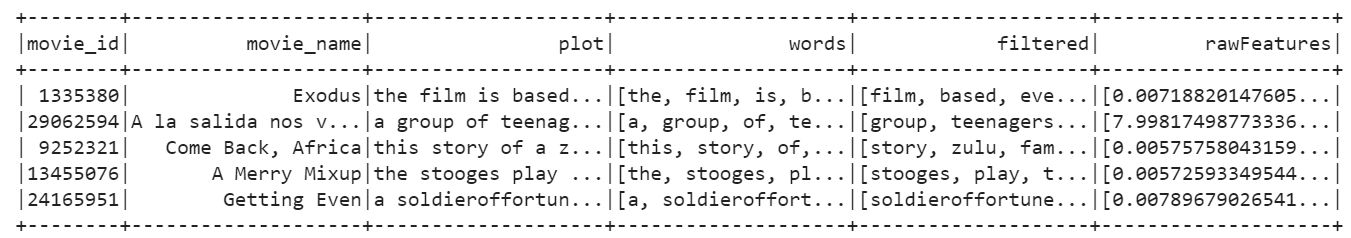
1. **Part 3: Custom Feature Engineering**

In this part we have implemented Word2Vec method. The Word2Vec model transforms each document into a vector using the average of all words in the document. We have chosen vectorSize as 100. Same method is used for train and test dataset.

The prepared **feature vectors for train** is as below:



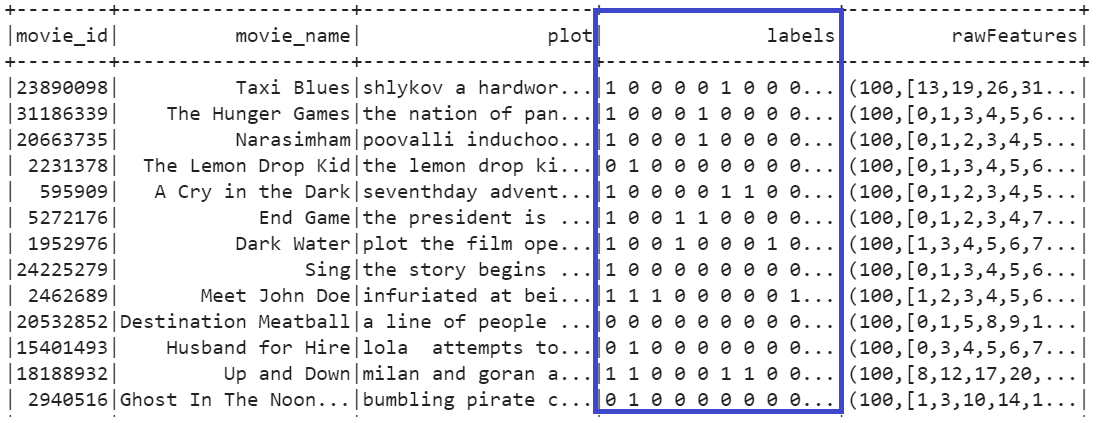
The prepared **feature vectors for test** is as below:



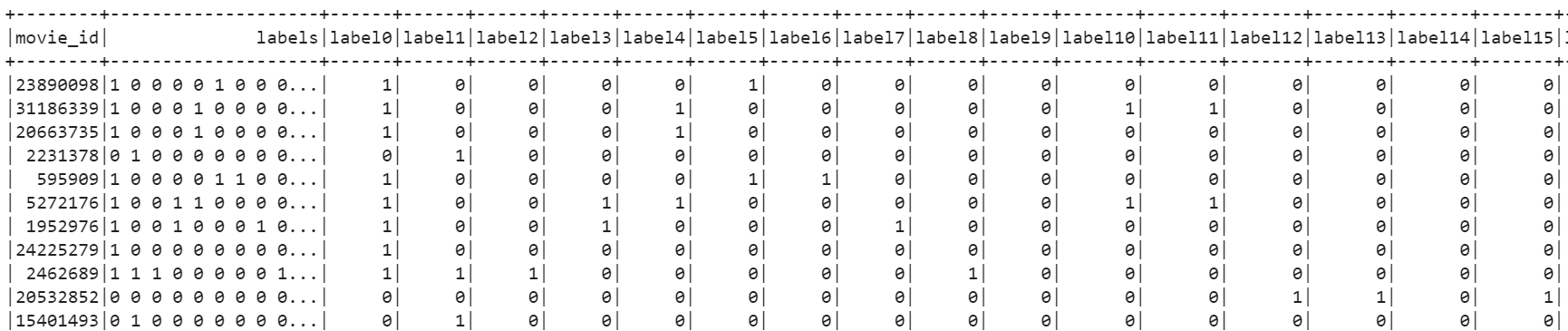
* **Label Processing:**

We have been given 20 unique labels and the corresponding index of the same. we will treat this multi-label classification problem as a Binary Relevance problem. Hence, we will now one hot encode the target variable.

After one-hot encoding the label column is looking like below:



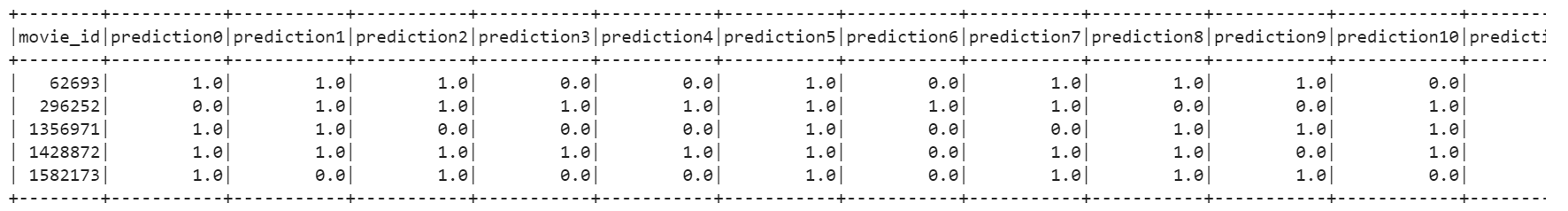
Now, we need to split the label column into 20 different columns for each category of the movie.



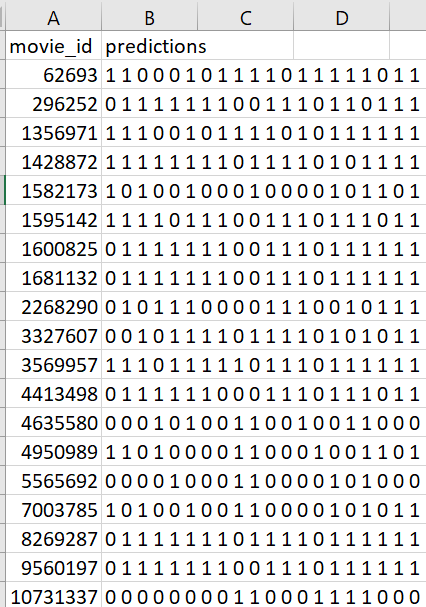
Next, we have combined feature vector with splitted label columns on the basis of movie id. So that we can pass this to the machine learning model classifier

* **Model Training & Evaluation:**

The next step is to put the features received from previous step into the classifier model. Here we have used RandomForest model. As our data is imbalanced, hence we have used undersampling to have better accuracy. As we need to predict for all 20 labels, we iterated over 20 times to predict the genres for each plot.



All 20 predictions have been merged into a single column and saved into a csv as follows:



* **Result:**

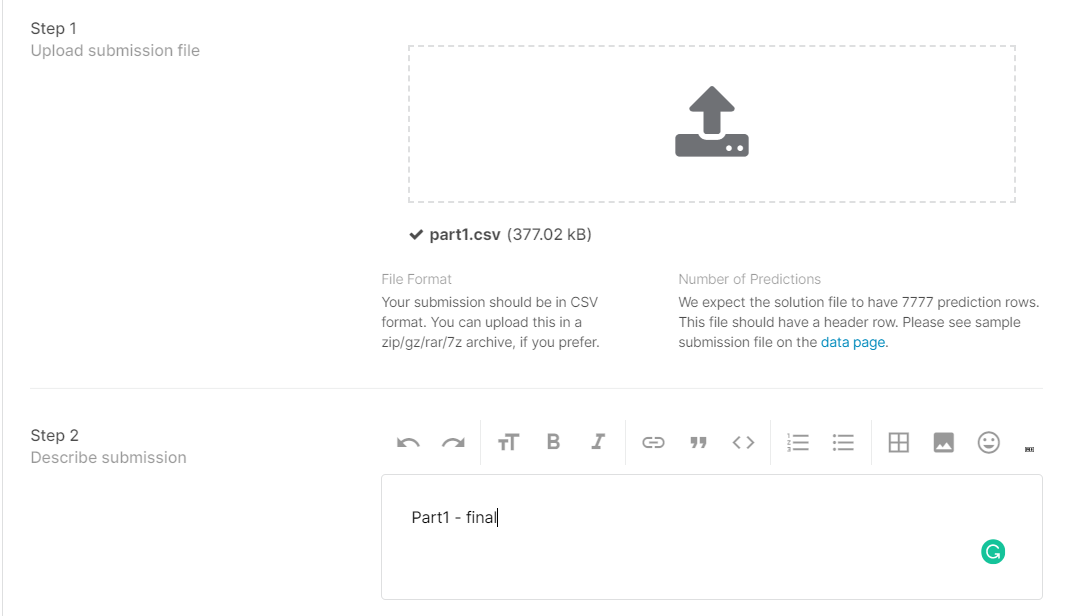
For all of the three parts generated prediction csv file has been uploaded into Kaggle to check the F1 score.

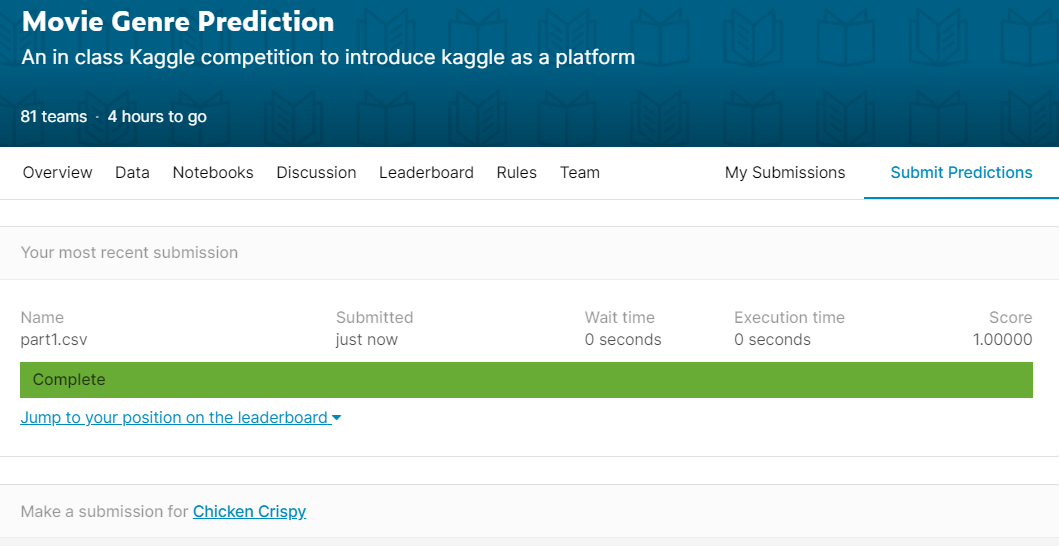
**1. Part 1: Basic Model**

Attached is the predictions from part 1:



We have pushed this csv file to Kaggle to get the F1 score



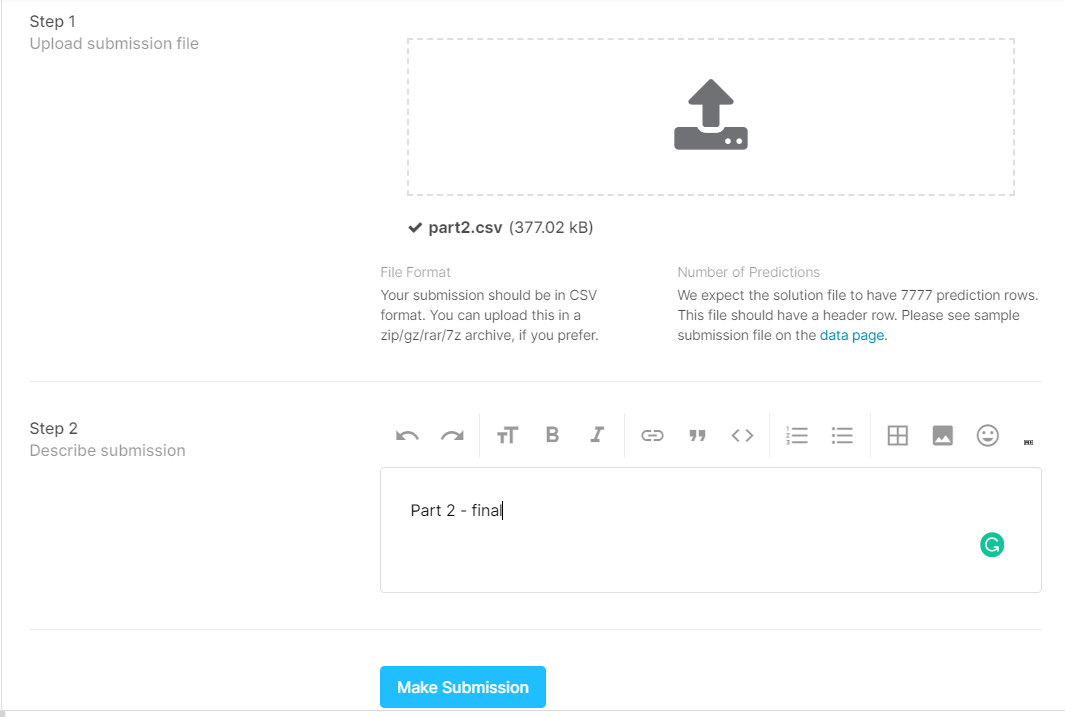


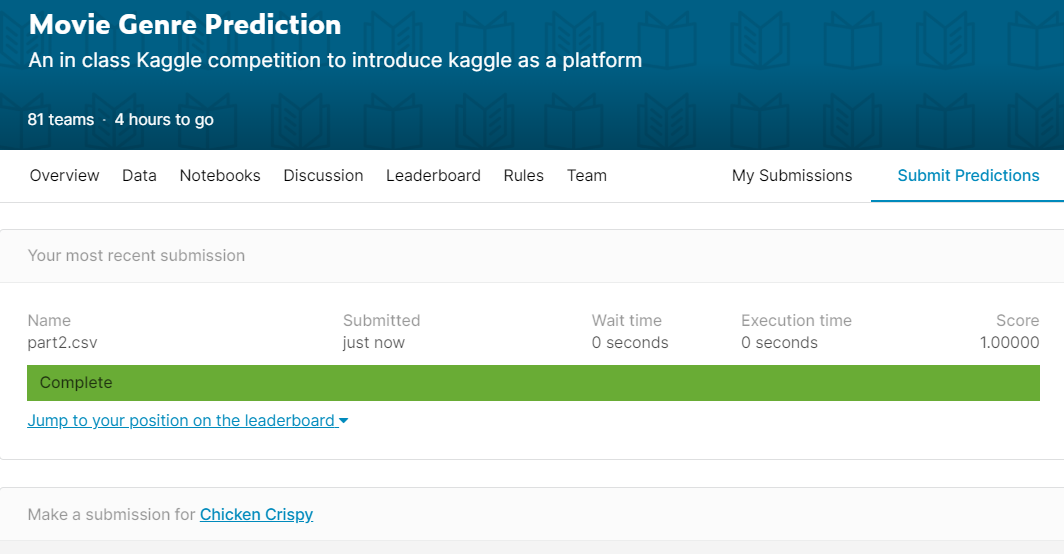
We have received the F1 score as 1.

1. **Part 2: Using TF-IDF**

Attached is the predictions from part 2:

****



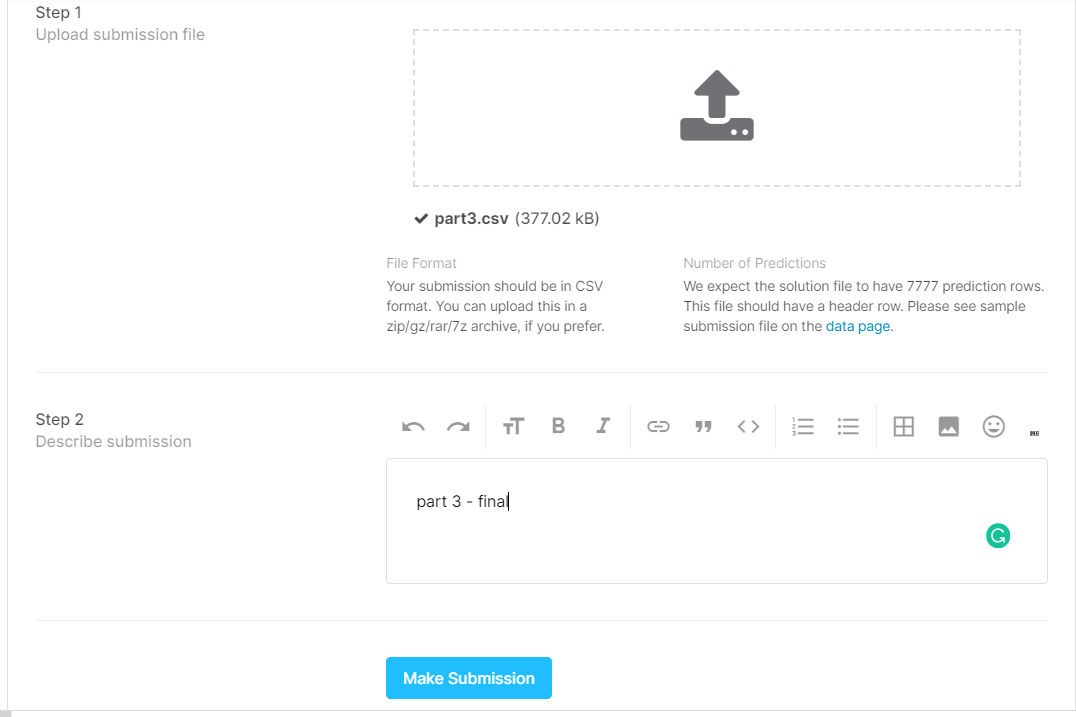


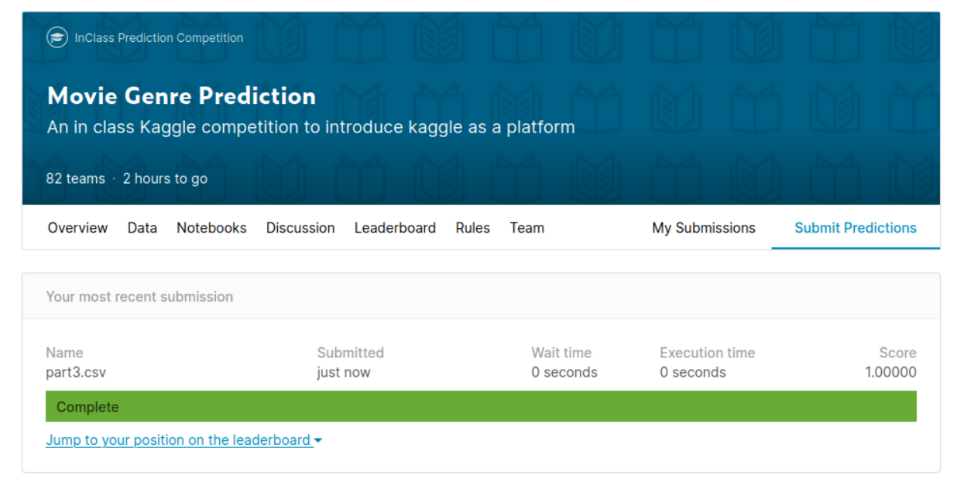
We have received the F1 score as 1.

1. **Part 3: Custom Feature Engineering**

Attached is the predictions from part 3:

****



****

We have received the F1 score as 1.

* **Conclusion:**

So, in this study we were able to implement the model to predict the movie genre based on the plot summaries successfully. All of the three parts have been completed with satisfactory F1 score.

* **Acknowledgments:**

We are extremely grateful to Professor Deen Dayal Mohan for teaching all the

necessary concepts related to Predictive Analytics, Apache **Spark** and helping in this project throughout. Also, we would like to express our sincere thanks and appreciation to Kyung Won Lee, Michael Long, Lawzeem, Chunwei Ma, Rui Li, Vinooth Rao Kulkarni to support us at every step where we find any difficulties.

* **References:**

<https://towardsdatascience.com/countvectorizer-hashingtf-e66f169e2d4e>

<https://spark.apache.org/docs/latest/ml-features#tokenizer>

<https://medium.com/towards-artificial-intelligence/multi-class-text-classification-using-pyspark-mllib-doc2vec-dbfcee5b39f2>

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

<https://towardsdatascience.com/multi-class-text-classification-with-pyspark-7d78d022ed35>

<https://datascienceplus.com/machine-learning-with-text-in-pyspark-part-1/>

<https://medium.com/@junwan01/oversampling-and-undersampling-with-pyspark-5dbc25cdf253>