

Movies Sentiment Analysis and

Classification Assignment

Submitted to

Vahid Hadavi

Submitted by:

Ramandeep Kaur- C0894968

Shubham Choksi-C0894399

Maaz Patel- C0892303

Zoyeb Mansuri-C0867190

Nagasai Velpuri – C0896277

INDEX

* Data Extraction---------------------------------------------------------------------------3
* Data Wrangling--------------------------------------------------------------------------4-5
  + Data cleaning
  + Data formatting
* Data Visualization-------------------------------------------------------------------------6
* Modelling the data------------------------------------------------------------------------6-7
* Results discussion and Conclusion-----------------------------------------------------7-8

**Data Extraction:**

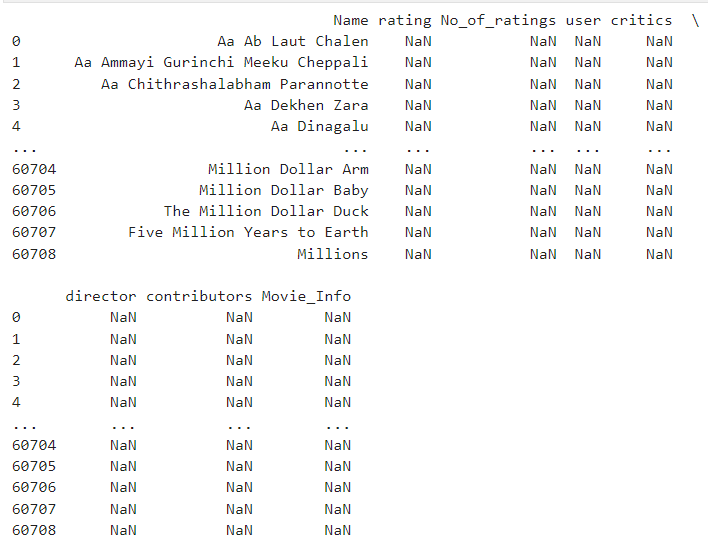
We are collecting the reviews of the Movies from the IMDB website. For this, we got a list of movies with around 60K records, out of which we picked the first 10K records and scrapped the reviews from the IMDB website. We use the “web driver” package from the selenium library to extract the data.

For data scrapping, first, we define the URL and use Webdriver.Chrome() we open the URL in Chrome browser and search for the movie name, and by using “CSS\_SELECTOR,” we are some elements from the list, namely ‘rating,’ ‘No\_of\_ratings,’ ‘user,’ ‘critics,’ ‘director,’ ‘contribution,’ and ‘Movie\_info.’ css\_selector picks the data by element and tag. The data might or might not be present in the global search of IMDB; hence, it is bound between the try-catch block(). After that, the whole data is stored in a data frame and converted into a CSV file, which is further used for analysis.

try:  
 driver.get(url)  
 search\_box = driver.find\_element(By.ID, 'suggestion-search')  
 driver.implicitly\_wait(5)  
 search\_box.send\_keys(df['Name'][i])  
 search\_box.send\_keys(Keys.RETURN)   
 driver.implicitly\_wait(5)  
 driver.find\_elements(By.CSS\_SELECTOR, "a.ipc-metadata-list-summary-item\_\_t")[0].click()  
 df.loc[i , 'rating'] = driver.find\_element(By.CSS\_SELECTOR, 'span.sc-bde20123-1.iZlgcd').text

Movie\_Info = ""  
 For j in storyLine:  
 Movie\_Info += j.text  
 Movie\_Info += ","  
 df.loc[i, 'Movie\_Info'] = Movie\_Info  
 print(df)  
except Exception as E:  
 clear\_output(wait=True)  
 continue

Sample Data after scrapping:



**Data Wrangling:**

After extracting the data, we might have some values and outliers to handle that need to be added.

As part of Data Wrangling, we need to do Data cleaning, Formatting, and Data Validation.

**Data Cleaning:**

As part of handling Null values, we are removing them as Null does not have much data and doesn’t contribute that much.

def remove\_null(self):  
 self.df = self.df.dropna()  
 self. shape = self.df.shape

We also have text data, so we need to clean that data. Firstly, we convert all the characters to lower, and then replace special characters and numbers in the data.

def apply\_lower\_case(self):  
 self.df[self.text\_column] = self.df[self.text\_column].apply(lambda string: string.lower())  
  
def clean\_text(self):  
 self.df[self.text\_column] = self.df[self.text\_column].apply(lambda string: re.sub(self.pattern, '', string))

**Data formatting:**

We have some numeric columns that are labelled short format. As part of Data formatting we are changing them to int format. No. of ratings, users and critics are classified and stored in rating\_popularity, user\_popularity and critic\_popularity respectively for easy of Analysis. After converting these into categories we are performing label encoder on these columns.

Null and integer handling:

def convert\_to\_int(value):  
 if "K" in value:  
 value = value.replace("K", "")  
 value = float(value) \* 1000  
 elif "M" in value:  
 value = value.replace("M", "")  
 value = float(value) \* 1000000  
 else:  
 pass  
 return int(value)  
  
 def remove\_null(self):  
 self.df = self.df.dropna()  
 self.shape = self.df.shape  
  
 def handle\_numeric\_vals(self):  
 columns = ["No\_of\_ratings", "user"]  
 for column in columns:  
 self.df[column] = self.df[column].apply(convert\_to\_int)

Encoding the categorical columns:

def apply\_descretization(self):  
 for column in self.numeric\_columns:  
 min = np.min(self.df[column])  
 max = np.max(self.df[column])  
 if column == "No\_of\_ratings":  
 bin\_edges = [min, 1000, 10000,100000,1000000,max ]  
 bin\_labels = ["Obscure", "Very Niche", "Emerging", "Popular", "Highly Popular"]  
 self.df['rating\_popularity'] = pd.cut(self.df[column], bins=bin\_edges, labels=bin\_labels)  
 if column == "user":  
 bin\_edges = [min, 25, 100, 500,1000, max]  
 bin\_labels = ["Mildly Positive", "Positive", "Very Positive", "Extremely Positive", "Exceptionally Positive"]  
 self.df["user\_popularity"] = pd.cut(self.df[column], bins = bin\_edges, labels = bin\_labels)  
 if column == "critics":  
 bin\_edges = [min, 25, 35, 70, 200, max ]  
 bin\_labels = ["Favorable", "Positive", "Highly Positive", "Exceptionally Positive", "Rave Reviews"]  
 self.df["critic\_popularity"] = pd.cut(self.df[column], bins = bin\_edges, labels = bin\_labels)  
  
def perform\_encoding(self):  
 columns = ['rating\_popularity', 'user\_popularity', "critic\_popularity"]  
 for column in columns:  
 self.df[column] = self.label\_encoder.fit\_transform(self.df[column])

For text data(Movies\_info) we are tokenizing the data and removing the stop words for better analysis. we are also applying the Lemmatization for the words. All this process we used methods for clean code and code practices.

Also the sentiment score is analysed using “score\_calculation.polarity\_scores” and stored in sentiment\_score column.

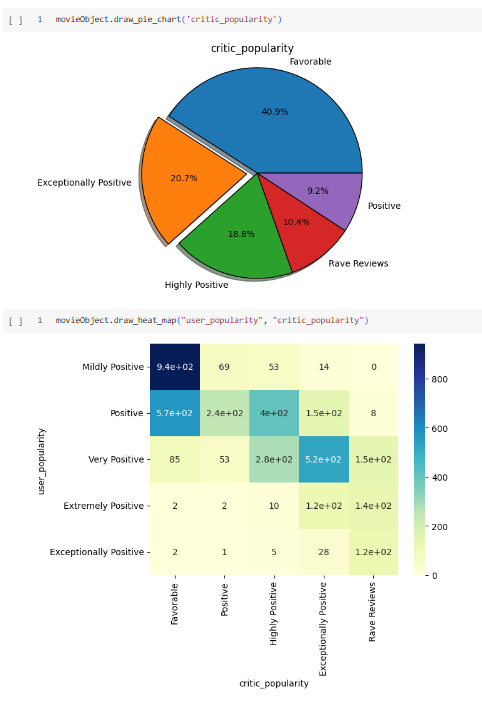
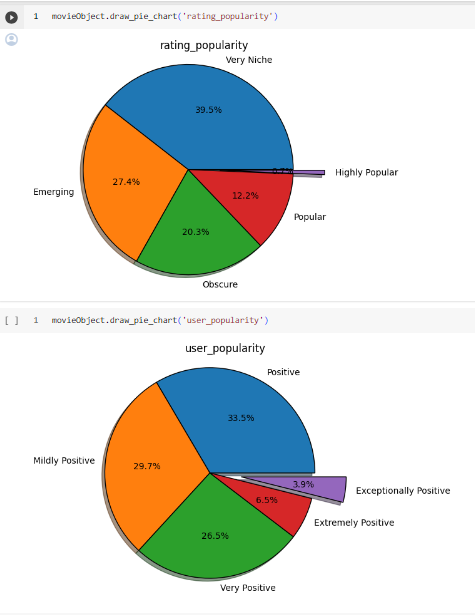
def tokenize(self):  
 self.df[self.text\_column] = self.df[self.text\_column].apply(lambda string: string.split())  
  
def handle\_stop\_words(self):  
 self.df[self.text\_column] = self.df[self.text\_column].apply(lambda string: [word for word in string if word not in self.stop\_words])  
  
def apply\_lematization(self):  
 self.df[self.text\_column] = self.df[self.text\_column].apply(lambda words: [self.lemmatizer.lemmatize(word) for word in words ])  
  
def get\_sentiment\_score(self):  
 self.df['Movie\_Info'] = self.df['Movie\_Info'].apply(lambda words: " ".join(words))  
 self.df['sentiment\_score'] = self.df['Movie\_Info'].apply(lambda string: self.score\_calculation.polarity\_scores(string)['compound'] )

**Data Visualization:**

For visualizing the data reviews we used pieChart from pyplot. Ater descritizing the values 'rating\_popularity', 'user\_popularity', 'critic\_popularity' we plotted them agaist number of ratings. We also have plotted the heat map for categories of ratings of user\_popularity and critic\_popularity.

In the rating\_popularity we see that less than 1% of the movies have the best rating possible while in the critic popularity “Extremely positive” category is achieved by about 20% of the movies. However, It seems that only4% of the movies are liked by the users as that is the range of the best figures.

From the heatmap we can validate and draw insight that movie that were marked “favourable” by the critics were marked as mildly positive by the users.



**Modeling the Data:**

For classifying the data, we have used two methods. Random forest classifier.

Firstly we have split the data into features X and target Y. We are predicting the user\_popularity based on rating\_popularity, critic\_popularity, sentiment\_score, and rating. We have divided the data into 80% for training and 20% for testing. We have given the neighbors parameter into 5.

After that, we are fitting the data into the classifier method.

def perform\_train\_test\_split(self):  
 X = self.df[['rating\_popularity', 'critic\_popularity', 'sentiment\_score', 'rating']]  
 y = self.df['user\_popularity']  
 self.X\_train, self.X\_test, self.y\_train, self.y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)  
  
def train\_classifier(self):  
 self.classifier.fit(self.X\_train, self.y\_train)  
  
def make\_prediction(self):  
 self.y\_pred = self.classifier.predict(self.X\_test)

**Random forest Classifier:**

For analyzing the parameters of RandomForest, we used the Grid search CV Hyperparameter tuning technique.

    # self.classifier.fit(self.X\_train, self.y\_train)

    param\_grid = {

        'n\_estimators': [50, 100, 200],

        'max\_depth': [None, 10, 20, 30],

        'min\_samples\_split': [2, 5, 10],

        'min\_samples\_leaf': [1, 2, 4],

        'bootstrap': [True, False]

    }

    grid\_search = GridSearchCV(estimator=self.classifier, param\_grid=param\_grid,

                           cv=3, n\_jobs=-1, verbose=2, scoring='accuracy')

    grid\_search.fit(self.X\_train, self.y\_train)

    # print("Best Parameters: ", grid\_search.best\_params\_)

    self.classifier = grid\_search.best\_estimator\_

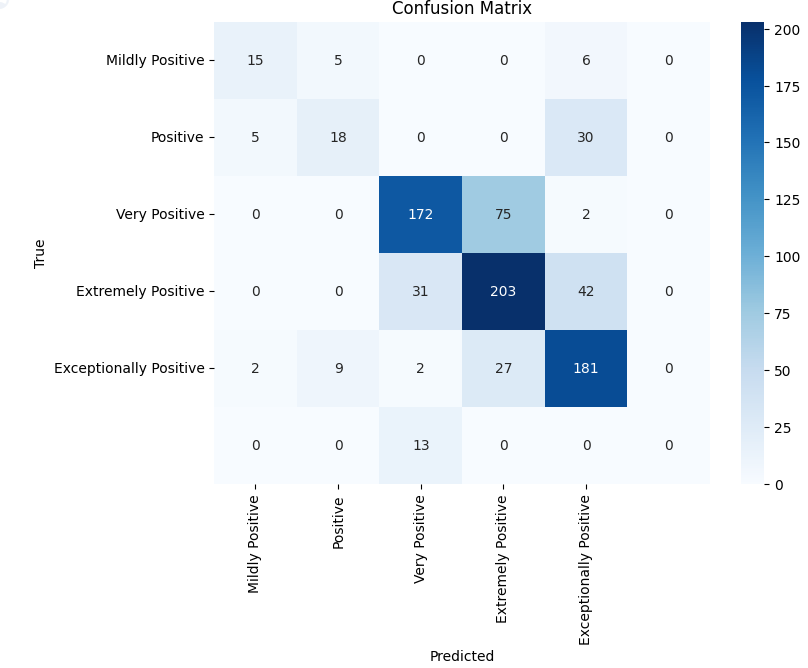
With these parameters, we are training the model using a Random forest algorithm with the same size, 80% for training and 20% for testing.

    self.classifier = RandomForestClassifier()

**Results Discussion and Conclusion:**

When it is performed using Random Forest, accuracy is 70.28%, and the confusion matrix for true v/s predicted is given below. From the confusion matrix, we can conclude the following:

* The classes Very positive, extremely positive, and exceptionally positive are predicted with the best accuracy.
* The extremely positive class is classified as the best. However, we also understand that about 30% of our classes in the dataset is “Extremely positive” and hence we can figure out that the model tends to be bias on that class.
* Furthermore, we can work on including movie genres in our dataet as they can provide valuable insights to our model.

.

Hence, a Random forest with the best grid parameters gives some accurate results in predicting.