Sentiment Analysis of IMDb's Avengers:Endgame Reviews

Meghana Atluri
Pavan Malapati
Naveen Kumar Kaparaju
Raja Vamsi Krishna Vadlamudi

How Many of you are Marvel fans here ?

Did you watch the Avengers : Endgame Movie ?

How was the Movie ?

OBJECTIVES

- **Analyze Sentiment**: Determine the overall sentiment of IMDb reviews (positive, negative) for Avengers Endgame.
- **Evaluate Models:** Compare the performance of machine learning models (Logistic Regression, SVM, Random Forest) in sentiment analysis.
- **Provide Insights**: Extract actionable insights from sentiment analysis to aid filmmakers, producers, and distributors.

REVIEWS

★ 10/10

Epic Ending
eandros-77451 26 April 2019

Thank you Marvel, End Game is an ending that give me speechless.

★ 8/10

Huge fan left underwhelmed!!!
james-84287 30 April 2019

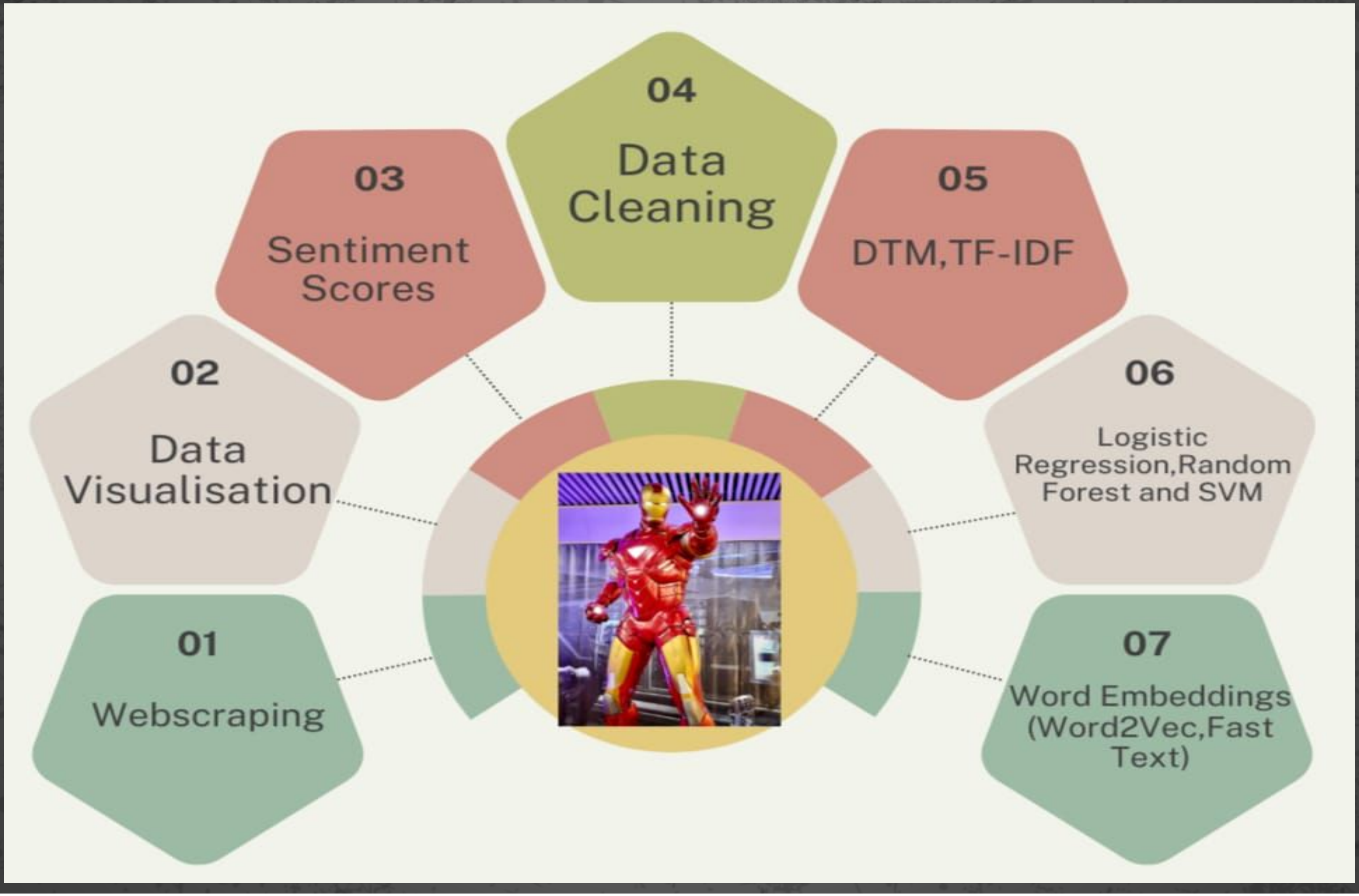
I went in with the highest expectations and came out underwhelmed and disappointed. its nowhere near as good as infinity war !!!

★ 1/10

One of the most overrated films of all time
moudekerk-82980 20 May 2019

Acting extremely sub-par, terrible story, and a cheap piece of non directional fan service.

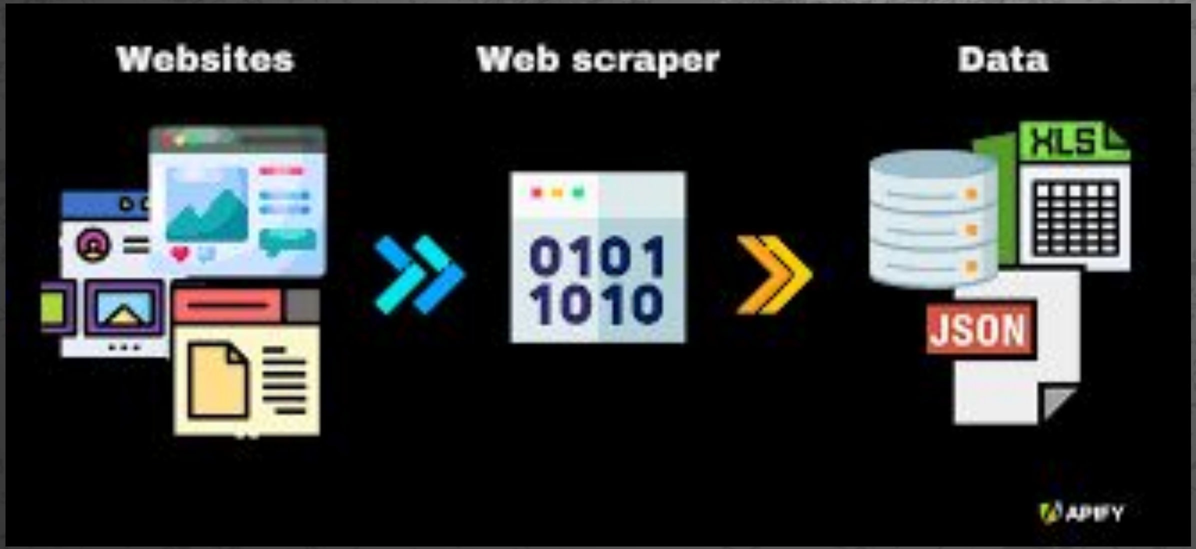
WORKFLOW



04
Data Cleaning
03
Sentiment Scores
05
DTM,TF-IDF
02
Data Visualisation
06
Logistic Regression,Random Forest and SVM
01
Webscraping
07
Word Embeddings (Word2Vec,Fast Text)

WEB SCRAPING

Web scraping is the automated process of extracting structured data from web pages, specifically designed for our project to gather IMDb reviews and ratings for the movie 'Avengers: Endgame' from the IMDb website.



1. Installing Python Libraries

```
[1]  !pip install selenium
     !pip install scrapy
     !pip install fake_useragent
```

2. Creating and Configuring Selenium WebDriver

```python
#all reviews and ratings
import pandas as pd
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# Set up Chrome WebDriver
chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--headless')
chrome_options.add_argument('--no-sandbox')
chrome_options.add_argument('--disable-dev-shm-usage')
driver = webdriver.Chrome(options=chrome_options)

# IMDb URL for Avengers: Endgame reviews
url = 'https://www.imdb.com/title/tt4154796/reviews?spoiler=hide&sort=curated&dir=desc&ratingFilter=0'
driver.get(url)
```

3. Function to click "Load More" button until all reviews are loaded

```python
# Wait for the reviews to load
wait = WebDriverWait(driver, 10)
wait.until(EC.presence_of_element_located((By.CLASS_NAME, 'lister-list')))

# Function to click "Load More" button until all reviews are loaded
def load_all_reviews():
    while True:
        try:
            load_more_button = driver.find_element(By.CLASS_NAME, 'ipl-load-more__button')
            driver.execute_script("arguments[0].scrollIntoView();", load_more_button)
            load_more_button.click()
            wait.until(EC.invisibility_of_element_located((By.CLASS_NAME, 'ipl-load-more__load-indicator')))
        except Exception as e:
            break
```

4. Extracting and Printing all reviews and ratings

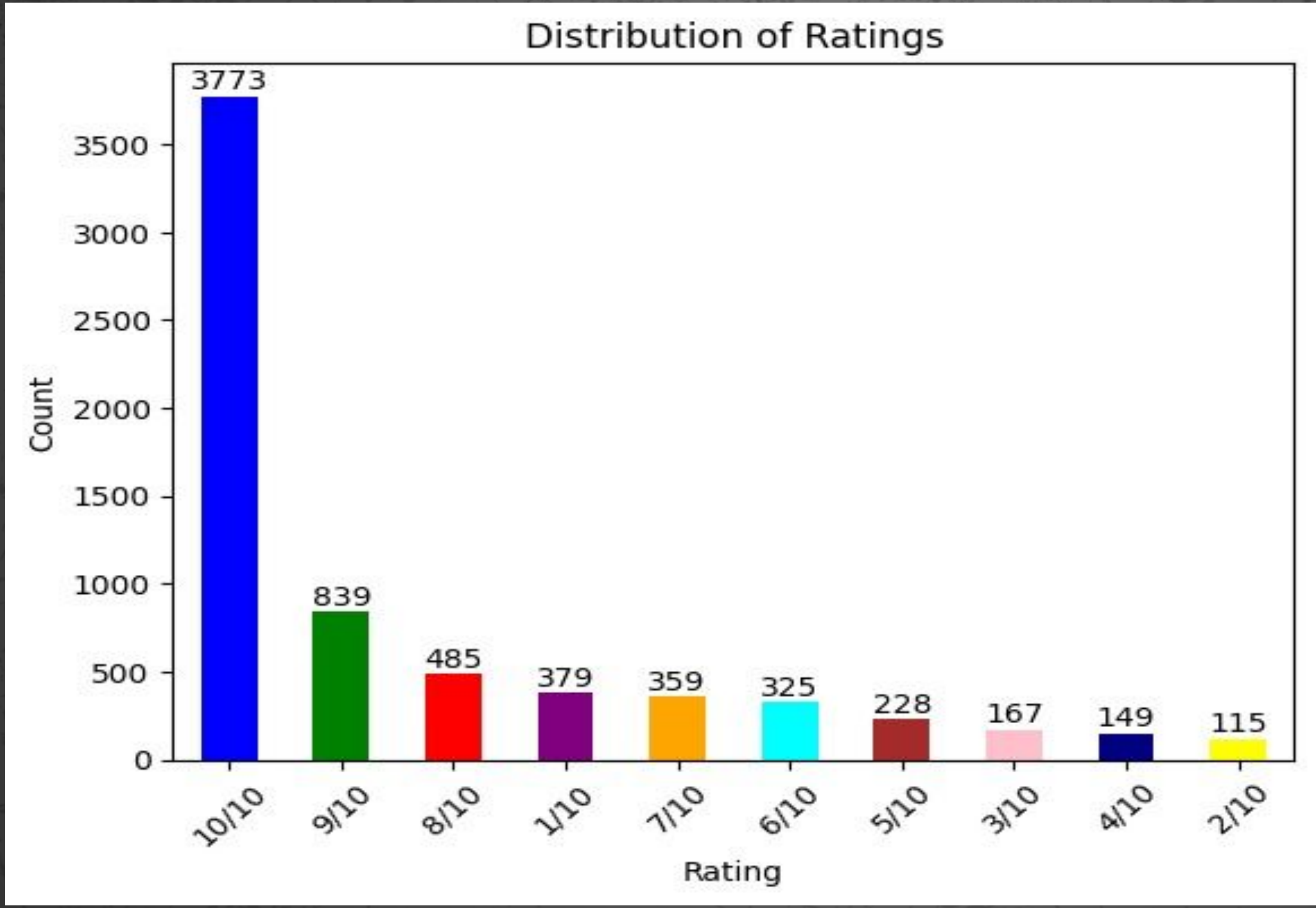
```python
# Function to extract ratings and reviews
def extract_data():
    rating_elements = driver.find_elements(By.CLASS_NAME, 'ipl-ratings-bar')
    review_elements = driver.find_elements(By.CLASS_NAME, 'text')

    data = []
    for rating_element, review_element in zip(rating_elements, review_elements):
        rating = rating_element.text.strip().split('\n')[0]
        review = review_element.text.strip()
        data.append({'Rating': rating if rating else None, 'Review': review if review else None})
        print(f"Rating: {rating if rating else 'None'}")
        print(f"Review: {review if review else 'None'}\n")
    return data
```

5. Saving the Data to Excel File

```
[ ]  # Extract text content from review elements
     from google.colab import drive
     drive.mount('/content/drive')

     import pandas as pd

     # Create a DataFrame to store the ratings and reviews
     ratings_reviews_df = pd.DataFrame(all_data)

     # Save DataFrame to Excel with ratings and reviews in different columns
     output_file = "/content/drive/MyDrive/Final Project/Reviews.xlsx"
     ratings_reviews_df.to_excel(output_file, index=False)
```

```
Mounted at /content/drive
  Rating                                             Review
0   7/10  But its a pretty good film. A bit of a mess in...
1  10/10  This film is an emotional rollercoaster with s...
2  10/10  First review from me. This film deserves it. A...
3  10/10  Thank you Marvel, End Game is an ending that g...
4  10/10  After watching Infinity war, I was looking for...
```

DATA VISUALISATION



WORD CLOUD

A word cloud is a visual representation of text data where words are displayed in varying sizes based on their frequency or importance within the text.

SENTIMENT SCORES

- Sentiment scores are numerical values(0,1) assigned to text data to quantify and categorize the sentiment or emotion expressed within the text.

- Sentiment scores can be calculated using tools like SentimentIntensityAnalyzer() and polarity_scores() from the NLTK package

- However, for our project,sentiment scores are derived from the rating given by users.

- Ratings greater than 7 are considered positive (1) and ratings less than or equal to 7 are considered negative (0).

```
[ ]  import pandas as pd

     # Load the data from the Excel file
     input_file = "/content/drive/MyDrive/Final Project/Reviews.xlsx"
     ratings_reviews_df = pd.read_excel(input_file)

     # Function to calculate sentiment score based on ratings
     def calculate_sentiment(rating):
         # Extract the numerical part of the rating (e.g., "7/10" -> 7)
         rating_value = float(rating.split('/')[0])
         # Calculate sentiment score
         sentiment_score = 1 if rating_value > 7 else 0
         return sentiment_score
```

	Rating	Review	Sentiment
0	7/10	But its a pretty good film. A bit of a mess in...	0
1	10/10	This film is an emotional rollercoaster with s...	1
2	10/10	First review from me. This film deserves it. A...	1
3	10/10	Thank you Marvel, End Game is an ending that g...	1
4	10/10	After watching Infinity war, I was looking for...	1
...	...	...	...
6814	5/10	Great expectations.. But just a good film. And...	0
6815	5/10	Ugh. This three hour opus did not need to be t...	0
6816	5/10	This movie is so horrible I don't know why Mar...	0
6817	5/10	I really don't get the hype around this mess. ...	0
6818	5/10	They spent a lot of time in this film trying t...	0

DATA LOADING

```
import pandas as pd

training = pd.read_excel("/content/drive/MyDrive/Final Project/Reviews-1S.xlsx", header=0)

y_train = training['Sentiment']
x_train = training.drop(["Sentiment","Rating"], axis=1)
```

- “y_train” represents the target variable containing ‘sentiment score’ column.

- “x_train” contains the input features ‘Review’ and ‘Sentiment score’ excluding the 'Rating' column which is just used for assigning sentiment scores.

DATA CLEANING

```
[ ]  # Data pre-processing

     import nltk
     import numpy as np
     import matplotlib.pyplot as plt

     # Import list of stopwords from library NLTK
     from nltk.corpus import stopwords
     nltk.download('stopwords')
```

1. Removing Stopwords

```
[ ]  # Remove Stopwords

     stopwords_list = stopwords.words("english")
     print(f'List of stopwords:\n{stopwords_list}\n')

     # We remove negation words in list of stopwords
     no_stopwords = ["not","don't",'aren','don','ain',"aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "has
                     'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wasn', "wasn't", 'weren
                     "won't", 'wouldn', "wouldn't"]
     for no_stopword in no_stopwords:
         stopwords_list.remove(no_stopword)

     print(f'Final list of stopwords:\n{stopwords_list}')
```

2. Lemmatize Reviews

```
# Lemmatize reviews

# Import Lemmatizer from NLTK
from nltk.stem import WordNetLemmatizer
nltk.download('wordnet')

lemmatizer = WordNetLemmatizer()

# function that receive a list of words and do lemmatization:
def lemma_stem_text(words_list):
    # Lemmatizer
    text = [lemmatizer.lemmatize(token.lower()) for token in words_list]
    text = [lemmatizer.lemmatize(token.lower(), "v") for token in text ]
    return text
```

```
#Negative Contractions

# create a function to change negative contractions
import re
re_negation = re.compile("n't ") # specify a pattern you want to find in a string

# function that receive a sequence of words and return the same sequence transforming
# abbreviated negations to the standard form.
def negation_abbreviated_to_standard(sent):
    sent = re_negation.sub(" not ", sent)
    return sent
```

3. Negative Contractions

Function for cleaning reviews

```python
def review_to_words(raw_review):
    # 1. Remove HTML tags
    review_text = BeautifulSoup(raw_review).get_text()

    # 2. Transform abbreviated negations to the standard form.
    review_text = negation_abbreviated_to_standard(review_text)

    # 3. Remove non-letters and non-numbers
    tokenizer = RegexpTokenizer(r'\w+')
    words = tokenizer.tokenize(review_text)

    # 4. Remove stop words
    meaningful_words = [w for w in words if w.lower() not in stopwords_list]

    # 5. Apply lemmatization function
    lemma_words = lemma_stem_text(meaningful_words)

    # 6. Join the words back into one string separated by space, and return the result.
    return( " ".join(lemma_words))

    # 7. Handling Short Words
    tokens = [word for word in tokens if len(word) >= 3]

    # 8. Handling Rare Words
    word_counts = Counter(tokens)
    common_words = set(word for word, count in word_counts.items() if count > 5)
    tokens = [word for word in tokens if word in common_words]

    # 9. # Drop rows with NaN values in the 'Reviews' column
    training.dropna(subset=['Review'], inplace=True)
```

DOCUMENT TERM MATRIX(DTM)

- A document-term matrix or term-document matrix is a matrix that describes the frequency of terms (words) that occur in a collection of documents.
- In a document-term matrix,rows correspond to documents in the collection, columns correspond to terms
- The CountVectorizer function(imported from sklearn library) creates a matrix representation of text data by counting the frequency of each word (unigrams and bigrams).

```
# Initialize CountVectorizer
c = CountVectorizer(stop_words='english', lowercase=True, token_pattern=r'\w+', ngram_range=(1, 2))
```

TF-IDF

- TFIDF measures the importance of a word by both term frequency and how many times the word appears in all documents.
- TFIDF is defined as the product of two statistics:term frequency tf(w,d) and inverse document frequency idf(w).

$$tf(w,d) = \frac{f_{w,d}}{\text{number of words in d}} = \frac{f_{w,d}}{\sum_{w'} f_{w',d}}$$

$$idf(w) = \log\left(\frac{N}{df(w)}\right), \quad tfidf(w,d) = tf(w,d) \times idf(w)$$

where f(w,d) is the frequency of word w in document d, N is the total number of documents and $df(w)$ is the number of documents that contain term w.

- The TfidfVectorizer function (imported from the sklearn library) transforms text data into a TF-IDF matrix representation, assigning weights to words based on their frequency in individual documents and their rarity across all documents. The parameters min_df=0.5 and max_df=0.90 control the inclusion of words based on their document frequency, while ngram_range = (1,2) specifies that both unigrams and bigrams should be considered in the vectorization process.

```
# Define some hyperparameters of encoded
vectorizer = TfidfVectorizer(min_df=0.5, max_df=0.90,ngram_range = (1,2))
```

Train-Test Split

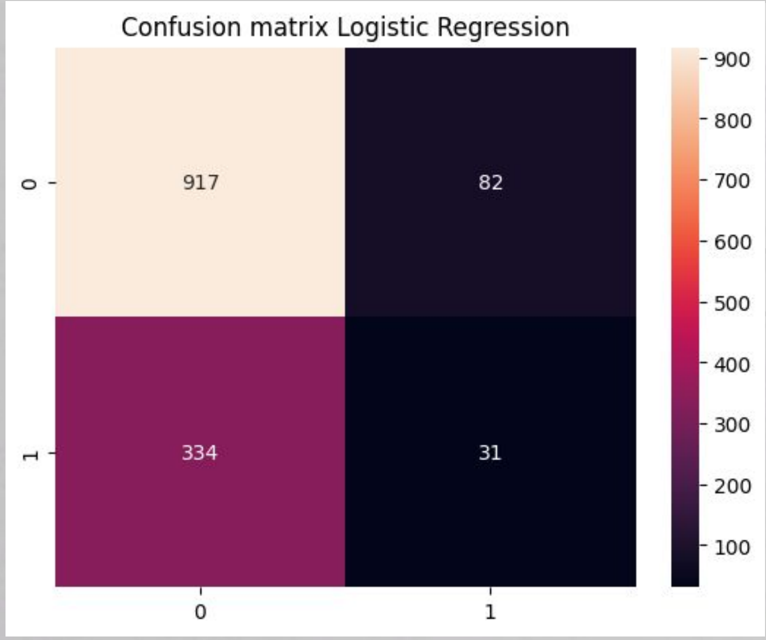
- Splitting the data into training and testing sets is a crucial step in machine learning model to assess the model's performance on unseen data; In our project, we divided into 80% training and 20% testing sets.

- The random_state parameter is set to 42 in the train_test_split function to ensure reproducibility, allowing for consistent results across different runs of the code.

LOGISTIC REGRESSION

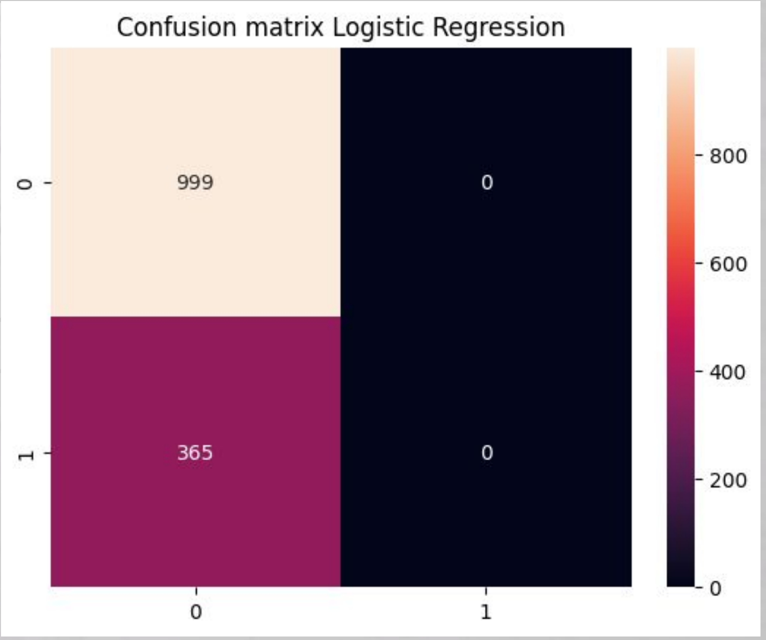
DTM

- **ACCURACY :** 0.7
- **PRECISION :** 0.733
- **RECALL SCORE :** 0.918
- **F1 SCORE :** 0.815



TF-IDF

- **ACCURACY :** 0.7416
- **PRECISION :** 0.754
- **RECALL SCORE :** 1
- **F1 SCORE :** 0.857



RANDOM FOREST

DTM

- **ACCURACY :** 0.73
- **PRECISION :** 0.732
- **RECALL SCORE :** 0.994
- **F1 SCORE :** 0.843

TF-IDF

- **ACCURACY :** 0.748
- **PRECISION :** 0.7612
- **RECALL SCORE :** 1
- **F1 SCORE :** 0.864

SVM

DTM

- **ACCURACY :** 0.731
- **PRECISION :** 0.732
- **RECALL SCORE :** 0.998
- **F1 SCORE :** 0.845

TF-IDF

- **ACCURACY :** 0.7425
- **PRECISION :** 0.757
- **RECALL SCORE :** 1
- **F1 SCORE :** 0.859

WORD EMBEDDINGS

Word2Vec :

- Word2Vec : method for generating word embeddings, its a dense vector representations of words that capture their semantic meanings. These embeddings can then be used as features in NLP tasks.
- In our project, we're using the skip-gram model [i.e given a target word, the model aims to predict the context words] within Word2Vec model.

- We then train Word2Vec model using gensim package with the following specifications: a vector size of 150 dimensions, a window size of 10 words, a minimum count of 3 occurrences for words, and employing 10 worker threads for efficient training.

```
model = gensim.models.Word2Vec(tokenized_reviews,vector_size=150,window=10,min_count=3,workers=10)
```

- We choose to train our Word2Vec model for 10 epochs.

```
model.train(tokenized_reviews, total_examples=len(tokenized_reviews), epochs=10)
```

```
similar_words = word2vec_model.wv.most_similar('great', topn=5)
print(similar_words)
```

```
[('fantastic', 0.9186002612113953), ('amaze', 0.9070403575897217), ('excellent', 0.9013640284538269), ('deliver', 0.8921491503715515), ('brilliant', 0.8832753300666809)]
```

```
[54] similar_words = word2vec_model.wv.most_similar('thanos', topn=5)
print(similar_words)
```

```
[('snap', 0.9181135892868042), ('brolin', 0.8936883211135864), ('josh', 0.8878936171531677), ('catastrophic', 0.8855810761451721), ('undo', 0.8682962656021118)]
```

The similarity score ranges between -1 and 1, where higher scores indicate greater similarity.

```
# Find the odd word out
odd_one_out = word2vec_model.wv.doesnt_match(["film", "thor", "ironman"])

# Print the odd word out
print(odd_one_out)
```

```
film
```

```
similarity_score = word2vec_model.wv.similarity(w1="love", w2="3000")

# Print the similarity score
print(similarity_score)
```

```
0.9523064
```

FastText :

- FastText, developed by Facebook AI Research (FAIR), is an extension of Word2Vec it is faster and also it enhance its efficiency and effectiveness by incorporating subword information and in handling out-of-vocabulary words..

- We created a FastText model, configuring it with a vector size of 100 dimensions, a window size of 5 words, a minimum count of 1 occurrence for words, and employing 4 worker threads for training efficiency.

```
model = FastText(sentences=tokenized_reviews, vector_size=100, window=5, min_count=1, workers=4)
```

- We choose to train our FastText model with 10 epochs.

```
#Training the model
model.train(sentences, total_examples=model.corpus_count, epochs=10)
```

```
[75] vocabulary = model.wv.index_to_key
     print('uniform' in vocabulary)

     False
```

```
vocabulary = model.wv.index_to_key
print('3000' in vocabulary)

True
```

```
[65] # Calculate similarity between two words
     similarity = model.wv.similarity("great", "amazing")
     print("Similarity between 'great' and 'amazing':", similarity)

     Similarity between 'great' and 'amazing': 0.90961784
```

```
[66] # Calculate similarity between two words
     similarity = model.wv.similarity("action", "adventure")
     print("Similarity between 'action' and 'adventure':", similarity)

     Similarity between 'action' and 'adventure': 0.6749082
```

Limitations

- The use of simple sentiment scoring (positive or negative) based solely on user ratings might overlook sentiments expressed in the reviews.
- The project's accuracy is influenced mainly due to unbalanced data, potentially leading to biased predictions.

FUTURE SCOPE

- Implementing Transformer models like BERT and GPT to enhance our sentiment analysis capabilities, leveraging their advanced language understanding for more accurate sentiment classification.

- Enhancing sentiment analysis to detect sarcasm, irony, or mixed feelings for a deeper understanding of user reviews.

- Expanding the project to analyze sentiments in multiple languages for a broader reach.

References

- https://www.imdb.com/title/tt4154796/reviews?spoiler=hide&sort=curated&dir=desc&ratingFilter=0
- https://kavita-ganesan.com/gensim-word2vec-tutorial-starter-code/
- https://www.analyticsvidhya.com/blog/2023/01/introduction-to-fasttext-embeddings-and-its-implication/


Thank You!