RECOMMENDATION SYSTEM FOR MOVIES

Final Project

Group 1

TOPIC OUTLINE

1. Dataset/EDA

2. Data pre-prepocessing

3. Confusion Matrix

4. Evaluation/conclusion

DATASET / EDA

DATASET

```python
from datasets import load_dataset
from transformers import pipeline
import pandas as pd

#Load the dataset
ds = load_dataset("cornell-movie-review-data/rotten_tomatoes", split = 'train')

# Show the first few rows of the dataset (optional, just for preview)
ds[:5]
```

Rotten Tomatoes is an American review-aggregation website for film and television

MODEL

```
[ ]  # Load sentiment analysis pipeline
     # Use a pipeline as a high-level helper

     #text-classification: performing text classification, including sentiment analysis
     #Model = pig4431/rtm_DistilBERT_5E is specific model use for this dataset
     pipe = pipeline("text-classification", model="pig4431/rtm_DistilBERT_5E")
```

DistilBERT - In a movie recommendation system, it can process movie descriptions, reviews, or metadata to create embeddings, enabling personalized suggestions by matching user preferences with similar movies efficiently.

DATA PRE-PROCESSING

DATA-CLEANING

```python
import re

def preprocess_text(text):
    text = text.lower()  # Convert to lowercase
    text = re.sub(r'[^a-zA-Z0-9\s]', '', text)  # Remove punctuation and special characters
    text = re.sub(r'\s+', ' ', text)  # Remove extra whitespace
    return text
```

```python
# Example of text preprocessing
def preprocess_text(text):
    # Lowercasing and other preprocessing if needed
    text = text.lower()  # Convert to lowercase
    return text

# Apply preprocessing to the dataset (if needed)
dataset = ds.map(lambda x: {'text': preprocess_text(x['text'])})
dataset[:5]
```

CHANGE TO LOWER CASE

STOPWORD REMOVAL

```python
#Remove stop words: ELiminate common words that don't contribute much meaning

import nltk
from nltk.corpus import stopwords
nltk.download('stopwords')


def remove_stopwords(text):
    stop_words = set(stopwords.words('english'))
    words = text.split()
    filtered_words = [word for word in words if word.lower() not in stop_words]
    return ' '.join(filtered_words)

# Apply stopword removal to the dataset
dataset = dataset.map(lambda example: {'text': remove_stopwords(example['text'])})
dataset[:5]
```

SPECIAL CHARACTER AND NUMBER REMOVAL

```python
#Remove special characters and numbers

import re

def remove_special_characters(text):
    # Remove special characters and numbers
    cleaned_text = re.sub(r'[^a-zA-Z\s]', '', text)
    return cleaned_text

# Apply cleaned text to dataset
dataset = dataset.map(lambda example: {'text': remove_special_characters(example['text'])})
dataset[:5]
```

STEMMING AND LEMMATIZATION

```python
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.tokenize import word_tokenize

# Download necessary NLTK resources if you haven't already
nltk.download('punkt')
nltk.download('wordnet')
nltk.download('punkt_tab')

def stem_and_lemmatize(text):
    words = word_tokenize(text)
    stemmer = PorterStemmer()
    stemmed_words = [stemmer.stem(word) for word in words]
    lemmatizer = WordNetLemmatizer()
    lemmatized_words = [lemmatizer.lemmatize(word) for word in stemmed_words]

    # Join the processed words back into a single string
    processed_text = ' '.join(lemmatized_words)

    return processed_text

# Apply the stem_and_lemmatize function to the 'text' column of your dataset
dataset = dataset.map(lambda example: {'text': stem_and_lemmatize(example['text'])})

# Display the first 5 examples of the processed dataset
dataset[:5]
```

```python
# Take a smaller sample of the dataset
sample_ds = ds[2000:6000]
sample_ds
```

“4000 ROWS”

```python
# Perform sentiment analysis on the sample
results = pipe(sample_ds['text'])

# Convert labels to "positive" or "negative"
for result in results:
    if result['label'] == 'LABEL_1':
        result['label'] = 'positive'
    elif result['label'] == 'LABEL_0':
        result['label'] = 'negative'
results
```

```python
# Assuming the dataset has a 'label' column with the true sentiment labels
true_labels = sample_ds['label']  # Actual labels

# Convert the true labels to match the model's label format (e.g., 0 = 'NEGATIVE', 1 = 'POSITIVE')
true_labels_mapped = ['POSITIVE' if label == 1 else 'NEGATIVE' for label in true_labels]

# Define the label_mapping dictionary (replace with your desired mapping)
label_mapping = {
    'positive': 'POSITIVE',
    'negative': 'NEGATIVE',
    # Add other mappings as needed
}

predicted_labels = [label_mapping.get(result['label'], result['label']) for result in results]

# Create a DataFr Loading... pare true and predicted labels
comparison_df = pd.DataFrame({
    'True Label': true_labels_mapped,
    'Predicted Label': predicted_labels,
    'Confidence Score': [result['score'] for result in results]
})
comparison_df
```

OUTPUT - >

	True Label	Predicted Label	Confidence Score
0	POSITIVE	POSITIVE	0.997387
1	POSITIVE	POSITIVE	0.997697
2	POSITIVE	POSITIVE	0.994891
3	POSITIVE	POSITIVE	0.995184
4	POSITIVE	POSITIVE	0.995725
...	...	...	...
3995	NEGATIVE	NEGATIVE	0.997764
3996	NEGATIVE	NEGATIVE	0.996827
3997	NEGATIVE	NEGATIVE	0.996943
3998	NEGATIVE	NEGATIVE	0.991760
3999	NEGATIVE	NEGATIVE	0.981436

4000 rows × 3 columns

```python
[ ] from sklearn.metrics import accuracy_score, confusion_matrix

    #Calculate the Accuracy

    accuracy = accuracy_score(true_labels_mapped, predicted_labels)
    print(f"Accuracy: {accuracy * 100:.2f}%")
```

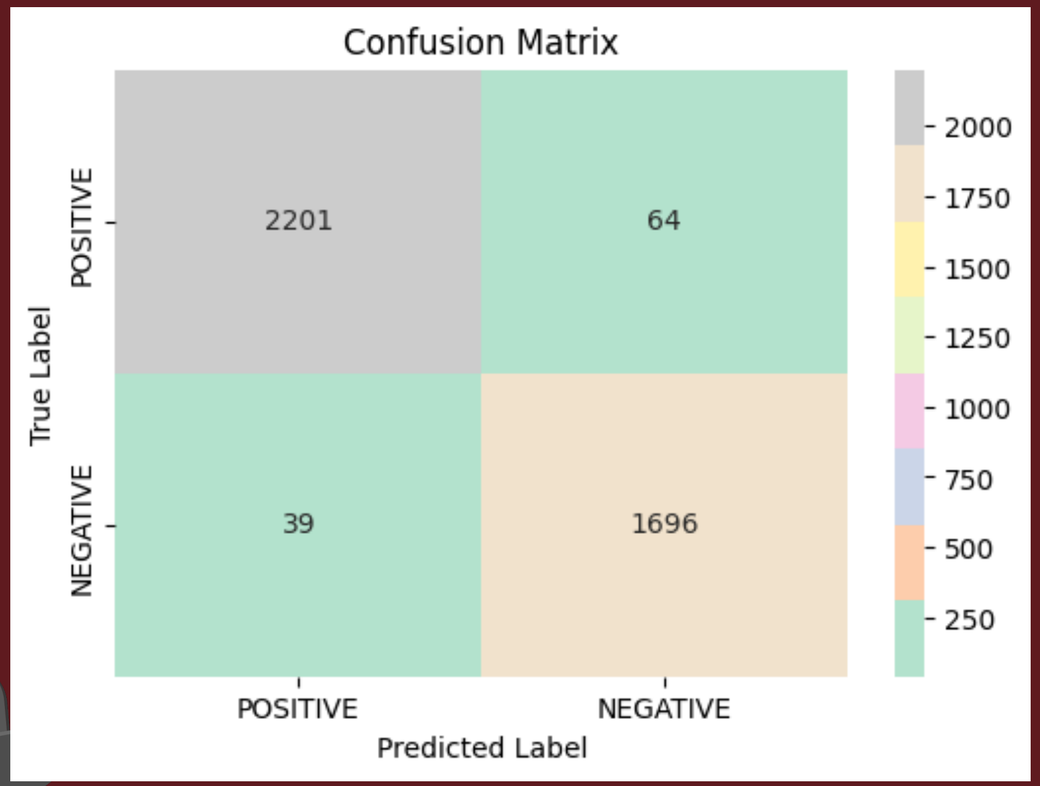
```
Accuracy: 97.42%
```

ACCURACY%

97.42%

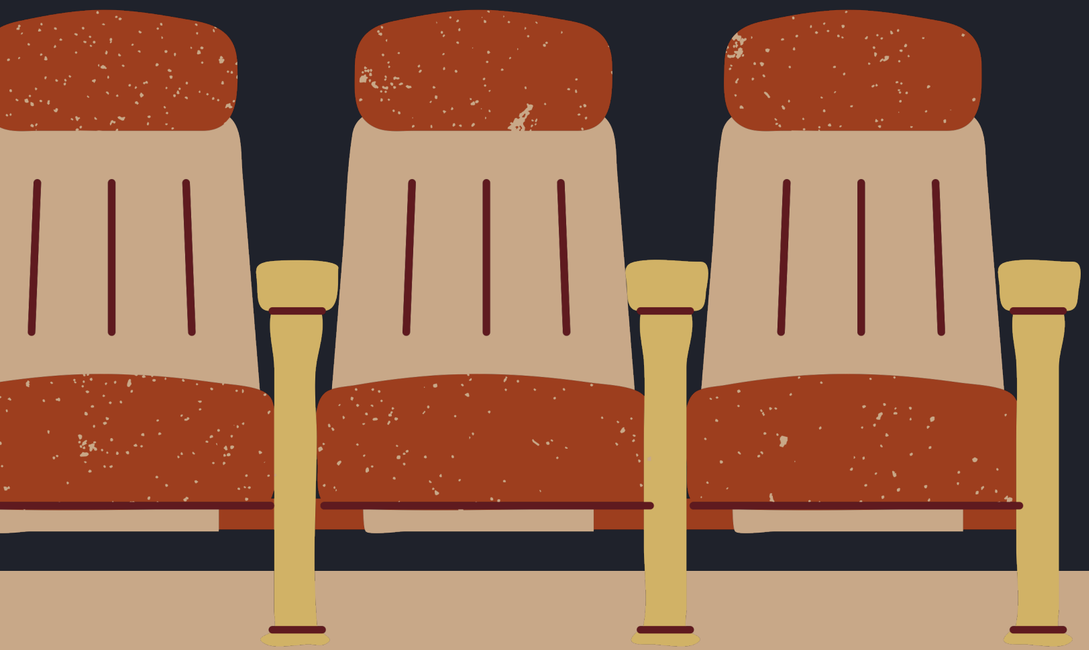
CONFUSION MATRIX

CONFUSION MATRIX



- **True Positive : 2201**
  - 2201 positive reviews were predicted as positive reviews
- **False Positive : 64**
  - 64 positive reviews were incorrectly predicted as negative reviews
- **False Negative : 39**
  - 39 negative reviews were predicted incorrectly as positive reviews
- **False Positive : 1696**
  - 1696 negative reviews were predicted correctly as negative reviews

EVALUATION



EVALUATION METRICS

Accuracy

```
[ ]  from sklearn.metrics import accuracy_score, confusion_matrix

     #Calculate the Accuracy

     accuracy = accuracy_score(true_labels_mapped, predicted_labels)
     print(f"Accuracy: {accuracy * 100:.2f}%")
```

```
Accuracy: 97.42%
```

Recall

```
#Calculate Recall

from sklearn.metrics import recall_score

recall = recall_score(true_labels_mapped, predicted_labels, pos_label='POSITIVE')
print(f"Recall: {recall * 100:.2f}%")
```

```
Recall: 97.17%
```

Precision

```
#Calculate Precision

from sklearn.metrics import precision_score

precision = precision_score(true_labels_mapped, predicted_labels, pos_label='POSITIVE')
print(f"Precision: {precision * 100:.2f}%")
```

```
Precision: 98.26%
```

PATTERN

2265
POSITIVE reviews in total
2201
were correctly predicted

1735
NEGATIVE reviews in total
1696
were correctly predicted

High numbers of true positive and negative value; therefore, well performance in detecting positive and negative reviews and **low misclassification rate**

DATASET BIASES

2265
POSITIVE reviews in total
2201
were correctly predicted

1735
NEGATIVE reviews in total
1696
were correctly predicted

The dataset shows a slight bias towards positive reviews, as there are more positive reviews (2,265) compared to negative ones (1,735). However, this minor imbalance is not a significant concern, as the evaluation metrics remain strong and indicate reliable performance.

CONCLUSION

- The **high** number of **accurately predicted positive and negative reviews** highlight the model's effectiveness in sentiment analysis.
- This indicates **strong performance** in detecting sentiments with **minimal misclassification**, supporting the model's reliability for practical applications.

THANK YOU FOR LISTENING

Don't hesitate to ask any questions!