Project Report: Emotion-Based Movie Recommendation System

Introduction

The aim of this project is to develop a movie recommendation system based on the emotions inferred from a user's input text. This involves the use of natural language processing (NLP) techniques to predict emotions from text and map these emotions to movie genres, followed by fetching top-rated movies in those genres from The Movie Database (TMDb). This project leverages machine learning models, specifically a fine-tuned DistilBERT model, to understand and classify emotions, and uses the TMDb API to fetch movie details.

Data Preparation and Exploration

Step 1: Dataset Download and Exploration

We begin by downloading the dataset which contains text samples labeled with different emotions. We explore the dataset to understand its structure and content.

```
import os
os.environ['kaggle_config_dir']='/content'
! kaggle datasets download -d nelgiriyewithana/emotions
! unzip /content/emotions.zip

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
import numpy as np
df=pd.read_csv('/content/text.csv')
```

Step 2: Data Cleaning and Preprocessing

We clean the dataset by removing unnecessary columns and mapping numerical labels to categorical emotions for better interpretability.

```
rating_to_emotion = {
    5: 'surprise',
    4: 'fear',
    3: 'angry',
    2: 'lovely',
    1: 'joy',
    0: 'sad'
}

df['label'] = df['label'].map(rating_to_emotion)
df = df.drop(columns='Unnamed: 0')
df = df.iloc[:200000]
```

Model Training

Step 3: Import Necessary Libraries

We import necessary libraries for data processing, model training, and evaluation.

```
!pip install transformers tensorflow

import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from transformers import DistilBertTokenizer,
TFDistilBertForSequenceClassification
from sklearn.preprocessing import LabelEncoder
```

Step 4: Data Encoding and Splitting

We encode the labels and split the dataset into training and testing sets.

```
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])

train_texts, test_texts, train_labels, test_labels = train_test_split(df['text'],
df['label'],test_size=0.2, random_state=42)
```

Step 5: Model Initialization

We initialize the DistilBERT tokenizer and model. DistilBERT is chosen for its balance between performance and computational efficiency.

```
tokenizer = DistilBertTokenizer.from_pretrained('distilbert-base-uncased')
model = TFDistilBertForSequenceClassification.from_pretrained
('distilbert-base-uncased', num_labels=len(label_encoder.classes_))
```

Step 6: Text Tokenization

We tokenize the text data for both training and testing sets.

```
train_encodings = tokenizer(list(train_texts.values), truncation=True, padding=True)
test_encodings = tokenizer(list(test_texts.values), truncation=True, padding=True)
```

Step 7: Dataset Preparation

We convert the tokenized data into TensorFlow dataset format and optimize the data pipeline.

```
train_dataset = tf.data.Dataset.from_tensor_slices((
    dict(train_encodings),
    train_labels
))

test_dataset = tf.data.Dataset.from_tensor_slices((
    dict(test_encodings),
    test_labels
))

AUTOTUNE = tf.data.AUTOTUNE

def prepare_dataset(dataset, batch_size, shuffle=False, cache=True):
    if cache:
        dataset = dataset.cache()
    if shuffle:
        dataset = dataset.shuffle(buffer_size=10000)
    dataset = dataset.batch(batch_size)
    dataset = dataset.prefetch(buffer_size=AUTOTUNE)
    return dataset

train_dataset = prepare_dataset(train_dataset, batch_size=32, shuffle=True)
test_dataset = prepare_dataset(test_dataset, batch_size=64)
```

Step 8: Model Compilation

We compile the model using the Adam optimizer and sparse categorical cross-entropy loss. Adam optimizer is chosen for its adaptive learning rate capabilities, which helps in faster convergence.

```
optimizer = tf.keras.optimizers.Adam(learning_rate=5e-5)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

model.compile(optimizer=optimizer, loss=loss, metrics=[metric])
```

Step 9: Model Training

We train the model for 3 epochs and evaluate its performance on the test set.

```
history = model.fit(train_dataset, epochs=3, validation_data=test_dataset)
eval_result = model.evaluate(test_dataset)
print(f"Test loss: {eval_result[0]}, Test accuracy: {eval_result[1]}")
```

Step 10: Model Saving

We save the fine-tuned model for future use.

```
model.save_pretrained('distilbert-finetuned-model')
tokenizer.save_pretrained('distilbert-finetuned-model')
```

Emotion Prediction and Attention Visualization

Step 11: Load Model and Tokenizer

We load the fine-tuned model and tokenizer for emotion prediction.

```
model_path = '/content/distilbert-finetuned-model'
tokenizer = DistilBertTokenizer.from_pretrained(model_path)
config = DistilBertConfig.from_pretrained(model_path, output_attentions=True)
model = TFDistilBertForSequenceClassification.from_pretrained
(model_path, config=config)
```

Step 12: Predict Emotion

We define a function to predict emotion from the input text and retrieve attention weights.

```
def predict_emotion_with_attention(text):
    inputs = tokenizer.encode_plus(
        text,
        add_special_tokens=True,
        max_length=128,
        padding='max_length',
        truncation=True,
        return_attention_mask=True,
        return_tensors='tf'
    )

    outputs = model(inputs)
    logits = outputs.logits
    probabilities = tf.nn.softmax(logits, axis=-1).numpy()[0]

    predicted_label = tf.argmax(probabilities, axis=-1).numpy()
    predicted_emotion = rating_to_emotion[predicted_label]
    attentions = outputs.attentions
    attention_weights = attentions[-1][0][0]

    return predicted_emotion, probabilities[predicted_label],
    attention_weights, inputs['input_ids']

text_input = "I feel really happy today!"
predicted_emotion, confidence, attention_weights,
input_ids = predict_emotion_with_attention(text_input)
tokens = tokenizer.convert_ids_to_tokens(input_ids[0])
```

Step 13: Attention Visualization

We visualize the attention weights to understand which words influenced the prediction.

```
def plot_attention(tokens, weights):
    fig, ax = plt.subplots(figsize=(10, 10))
    ax.matshow(weights, cmap='viridis')

    ax.set_xticks(range(len(tokens)))
    ax.set_yticks(range(len(tokens)))

    ax.set_xticklabels(tokens, rotation=90)
    ax.set_yticklabels(tokens)

    plt.show()

plot_attention(tokens, attention_weights.numpy())
print(f"Predicted Emotion: {predicted_emotion}")
print(f"Confidence: {confidence}")
```

Movie Recommendation

Step 14: Fetch Movies from TMDb

We define a function to fetch top movies from TMDb based on genres and language.

```
def get_top_movies_tmdb(genres, api_key, language, count):
    genre_ids = {
        'Action': 28,
        'Adventure': 12,
        'Animation': 16,
        'Biography': 36,
        'Comedy': 35,
        'Crime': 80,
        'Drama': 18,
        'Family': 10751,
        'Fantasy': 14,
        'History': 36,
        'Horror': 27,
        'Musical': 10402,
        'Mystery': 9648,
        'Romance': 10749,
        'Sci-Fi': 878,
        'Thriller': 53,
        'War': 10752,
        'Western': 37
    }

    top_movies = []
    for genre in genres:
        genre_id = genre_ids.get(genre)
        if genre_id is not None:
            response = requests.get(f"https://api.themoviedb.org
            /3/discover/movie?api_key={api_key}&with_genres={genre_id}&with_original
            data = response.json()
            if 'results' in data:
                for movie in data['results']:
                    movie_id = movie['id']
                    movie_details=requests.get
                    (f"https://api.themoviedb.org/3/movie
                    /{movie_id}?api_key={api_key}&append_to_response=credits,videos"
                    .json()
                    cast = [cast_member['name'] for cast_member in movie_details['cr
                    trailer_link = None
                    for video in movie_details
                    ['videos']['results']:
                        if video['type'] ==
                        'Trailer' and video['site'] == 'YouTube':
                            trailer_link = f"https://www.youtube.com/watch?v={video[
                            break
                    duration = movie_details.get('runtime', 'N/A')
                    top_movies.append({
                        'title': movie['title'],
                        'description': movie.get('overview',
                        'No description available.'),
                        % Placeholder for remaining movie
                        details (continued in actual implementation)
```