**MOVIE REVIEW PREDICTION**

**1. Project Content**

This project implements a sentiment analysis model to classify movie reviews as either positive ("Hit 🎬") or negative ("Flop 💣") using the IMDb dataset. It contains:

- Data preprocessing with tokenization and padding sequences.

- Building and training a neural network model with Keras embedding layers.

- Saving and loading the trained model for reuse.

- A function to convert raw review text into model-compatible sequences.

- A prediction function that outputs a sentiment label based on the model's prediction.

- A simple command-line interface for users to input their own movie reviews and get sentiment predictions.

**2. Project Code**

from tensorflow.keras.preprocessing.sequence import pad\_sequences

from tensorflow.keras.datasets import imdb

from tensorflow.keras.models import load\_model

from tensorflow.keras.preprocessing.text import Tokenizer

import numpy as np

# Load IMDb word index and adjust indices to account for reserved tokens

word\_index = imdb.get\_word\_index()

index\_word = {v+3: k for k, v in word\_index.items()}

word\_index = {k: (v+3) for k, v in word\_index.items()}

word\_index["<PAD>"] = 0

word\_index["<START>"] = 1

word\_index["<UNK>"] = 2

word\_index["<UNUSED>"] = 3

# Load IMDb dataset limited to top 10,000 words

(x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=10000)

# Pad sequences to a fixed length (256)

x\_train = pad\_sequences(x\_train, maxlen=256, padding='post')

x\_test = pad\_sequences(x\_test, maxlen=256, padding='post')

# Build the model

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Embedding, GlobalAveragePooling1D, Dense

model = Sequential([

Embedding(10000, 16),

GlobalAveragePooling1D(),

Dense(16, activation='relu'),

Dense(1, activation='sigmoid')

])

# Compile the model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train the model (2 epochs for demo; increase epochs for better accuracy)

model.fit(x\_train, y\_train, epochs=2, batch\_size=512, validation\_split=0.2)

# Save the trained model

model.save("movie\_review\_model.h5")

# Load the saved model (use this block after initial training)

model = load\_model("movie\_review\_model.h5")

# Function to convert raw review text into padded sequence of integers

def review\_to\_sequence(text):

tokens = text.lower().split()

seq = [word\_index.get(word, 2) for word in tokens] # 2 is <UNK> for unknown words

return pad\_sequences([seq], maxlen=256, padding='post')

# Prediction function

def predict\_review(text):

sequence = review\_to\_sequence(text)

prediction = model.predict(sequence)[0][0]

return "Hit 🎬" if prediction > 0.5 else "Flop 💣"

# Command-line interface for quick testing

if \_\_name\_\_ == "\_\_main\_\_":

review = input("Enter a movie review: ")

result = predict\_review(review)

print("Prediction:", result)

**3. Key Technologies**

* **Python 3**: Primary programming language for model building and scripting.
* **TensorFlow & Keras**: Deep learning framework used for building, training, and saving the neural network.
* **IMDb Dataset**: A publicly available movie review dataset containing 50,000 reviews labeled positive or negative.
* **NumPy**: For numerical operations on sequences and arrays.
* **Sequence Padding**: Ensures input data is of uniform length for neural network compatibility.
* **Embedding Layer**: Converts words into dense vectors that capture semantic meaning.
* **Dense Layers**: Fully connected layers for learning nonlinear classification boundaries.

**4. Description**

**Dataset**

The project utilizes the IMDb movie reviews dataset, which contains:

* **50,000 reviews**: Balanced dataset with 25,000 positive and 25,000 negative reviews.
* **Pre-tokenized data**: Words converted to integers representing the top 10,000 most frequent words.

**Data Preprocessing**

* **Word Index Adjustment**: The dataset reserves indices 0-3 for special tokens such as <PAD>, <START>, <UNK>, and <UNUSED>.
* **Padding**: Each review is padded or truncated to 256 words to maintain consistent input size.
* **Unknown Word Handling**: Words not found in the vocabulary are replaced with the <UNK> token index (2).

**Model Architecture**

* **Embedding Layer**: Maps each word index to a 16-dimensional vector.
* **Global Average Pooling**: Collapses the sequence dimension by taking the average, reducing complexity.
* **Dense Layers**:
  + First dense layer with 16 neurons and ReLU activation.
  + Output layer with 1 neuron and sigmoid activation for binary classification.

**Training**

* Model is trained for 2 epochs using Adam optimizer and binary cross-entropy loss.
* Validation split of 20% helps monitor overfitting.

**Prediction Workflow**

* New reviews entered as raw text are tokenized and converted to sequences using the saved word index.
* Sequences are padded to 256 tokens.
* Model outputs a probability between 0 and 1.
* Threshold 0.5 used to classify sentiment into "Hit 🎬" (positive) or "Flop 💣" (negative).

**5. Output**

Epoch 1/2

40/40 [==============================] - 4s 72ms/step - loss: 0.4920 - accuracy: 0.7817 - val\_loss: 0.3810 - val\_accuracy: 0.8528

Epoch 2/2

40/40 [==============================] - 3s 66ms/step - loss: 0.3315 - accuracy: 0.8683 - val\_loss: 0.3399 - val\_accuracy: 0.8602

**Sample Predictions**

| **Review Text** | **Model Prediction** |
| --- | --- |
| "This movie was fantastic! The performances were Oscar-worthy." | Hit 🎬 |
| "Terrible film. Waste of time and money." | Flop 💣 |
| "Mediocre plot but good acting." | Flop 💣 (close) |
| "Loved the cinematography and soundtrack." | Hit 🎬 |

**Example**

Enter a movie review: The story was engaging and the acting superb.

Prediction: Hit 🎬

**6. Further Research**

**Improvements & Extensions**

* **Increase Training Epochs**: Train for more epochs to improve accuracy and reduce underfitting.
* **Use Pretrained Word Embeddings**: Replace Keras embedding with pretrained GloVe or Word2Vec embeddings for better semantic understanding.
* **Hyperparameter Tuning**: Experiment with embedding size, number of dense layers, dropout, batch size, and optimizer parameters.
* **Bidirectional RNNs / LSTMs**: Use recurrent architectures to capture word order and context better.
* **Attention Mechanisms**: Incorporate attention layers for improved focus on important words/phrases.
* **Fine-tune Tokenization**: Use advanced tokenizers like tensorflow\_text or spaCy for better text normalization and subword tokenization.
* **Multi-class Sentiment**: Extend the model to classify multiple sentiment levels (e.g., very positive, neutral, negative).
* **Real-time Web App**: Deploy using Flask, FastAPI, or Gradio for interactive user interface.
* **Explainability**: Integrate LIME or SHAP to explain predictions, helping build user trust.
* **Multilingual Support**: Expand dataset and model to handle reviews in multiple languages.