A MINI PROJECT REPORT ON

SENTIMENT ANALYSIS ON IMDB REVIEWS

Submitted by

LITHESH M (231501084)

PRATHISHA R (231501119)

AD23632 FRAMEWORK FOR DATA AND VISUAL ANALYTICS

Department of Artificial Intelligence and Machine Learning

Rajalakshmi Engineering College, Thandalam

BONAFIDE CERTIFICATE

NAME ……..……..…

ACADEMIC YEAR……..……SEMESTER……..BRANCH……

UNIVERSITY REGISTER No. | |

Certified that this is the bonafide record of work done by the above students on the Mini Project titled **"SENTIMENT ANALYSIS ON IMDB REVIEWS"** in the subject **AD23632 FRAMEWORK FOR DATA AND VISUAL ANALYTICS** during the year **2025 - 2026.**

Signature of Faculty – in – Charge

Submitted for the Practical Examination held on _____

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ABSTRACT

This project presents an intelligent system for classifying the sentiment of IMDB movie reviews as either positive or negative. The system leverages deep learning, specifically a Long Short- Term Memory (LSTM) recurrent neural network, to understand the context and sequential nature of text data. Utilizing the TensorFlow and Keras libraries, the model is built and trained on the standard IMDB dataset, which contains 25,000 movie reviews for training. The process involves tokenizing the text reviews, converting them to sequences, and applying padding to ensure uniform input length (maxlen=200). The core of the system is a sequential model composed of an Embedding layer, an LSTM layer, and Dense layers, culminating in a sigmoid activation function to output a sentiment probability. The system's effectiveness is demonstrated by a test accuracy of 84.04% and an AUC (Area Under the Curve) of 0.90, indicating strong classification performance. Additionally, a web application is developed using Flask to deploy the trained model. This API receives raw text reviews, preprocesses them using a saved Tokenizer, and returns the predicted sentiment in real-time. This solution provides an efficient and accurate method for automated opinion mining, which is highly valuable for market research, social media monitoring, and customer feedback analysis.

Keywords:

Sentiment Analysis, Deep Learning, LSTM, Natural Language Processing (NLP), IMDB Dataset, Keras, TensorFlow, Flask API.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In the modern digital economy, user-generated content is one of the most valuable resources available. E-commerce sites, social media platforms, and movie review aggregators like IMDB generate millions of text-based opinions daily. This data represents a "gold mine" of public opinion, offering direct insights into customer satisfaction, product reception, and public sentiment. However, the sheer volume and velocity of this data make it impossible to analyze manually.

This challenge has given rise to the field of Sentiment Analysis, a subfield of Natural Language Processing (NLP) concerned with the automated identification and extraction of opinions, emotions, and attitudes from text. Businesses use it to understand brand perception, movie studios to predict box office success, and service providers to identify areas for improvement.

Traditional methods for sentiment analysis, such as keyword spotting or early machine learning models (e.g., Naive Bayes), have proven brittle. They fail to capture the complex nuances of human language, such as context, sarcasm, and negation. For example, the phrases "this movie was not good" and "this movie was good" are fundamentally different, but a simple model might misinterpret them.

This is where Deep Learning offers a transformative solution. Models like Recurrent Neural Networks (RNNs) and, specifically, Long Short-Term Memory (LSTM) networks are designed to process sequential data. They

maintain an internal "memory" that allows them to understand the context of a word based on the words that came, even over long sentences. This project harnesses the power of LSTMs to build a robust and accurate sentiment classifier for movie reviews.

1.2 PROBLEM STATEMENT

To design, train, and deploy a deep learning model capable of accurately classifying the sentiment of a given text (a movie review) as either Positive or Negative. The system must handle the complexities of natural language and be exposed via a simple web API for real-time predictions.

1.3 OBJECTIVES

The primary objectives of this mini-project are:

1. To understand and implement a deep learning pipeline for a Natural Language Processing task.
2. To load and preprocess a large-scale text dataset (IMDB) for use with a neural network.
3. To build, train, and evaluate an LSTM-based sequential model for binary sentiment classification using TensorFlow and Keras.
4. To analyze the model's performance using standard metrics like accuracy and the ROC- AUC curve.
5. To deploy the trained model as a lightweight, real-time web service using the Flask framework.

1.4 REPORT STRUCTURE

This report is organized into the following chapters:

- Chapter 1: Introduction provides the background, motivation, problem statement, and objectives for the project.
- Chapter 2: Literature Review discusses existing work in the field of sentiment analysis, from traditional methods to modern deep learning approaches.
- Chapter 3: System Requirements lists the hardware and software components necessary to replicate this project.
- Chapter 4: System Overview details the existing systems, their drawbacks, and the architecture of our proposed system.
- Chapter 5: System Implementation provides a detailed, technical breakdown of the project's architecture, data flow, and modules.
- Chapter 6: Result and Discussion presents the performance of the trained model, including accuracy, ROC curves, and an analysis of the results.
- Chapter 7: Conclusion and Future Scope summarizes the project's achievements and suggests potential areas for future improvement.
- Appendix contains the complete source code for training and deploying the model.
- References lists the academic and technical sources cited in this report.

CHAPTER 2

LITERATURE REVIEW

2.1 OVERVIEW OF SENTIMENT ANALYSIS

Sentiment Analysis, or opinion mining, has been a topic of research for decades. The task's core difficulty lies in the ambiguity and context-dependent nature of human language. Early approaches, as reviewed by Pang and Lee (2008), focused on lexicon-based methods, which used dictionaries of "positive" and "negative" words. While simple, these methods are easily confused by negation ("not bad"), sarcasm, and domain-specific language.

2.2 TRADITIONAL MACHINE LEARNING APPROACHES

[1] Title: Machine Learning Approaches for Sentiment Classification

This study provides a comparative analysis of traditional machine learning algorithms for sentiment analysis, specifically Naive Bayes, Support Vector Machines (SVM), and Logistic Regression. Using a standard movie review dataset, the study found that SVM achieved the highest accuracy at 87%. However, a significant limitation was the model's reliance on feature engineering (e.g., TF-IDF) and its inability to effectively process sarcasm and complex sentence structures involving negation.

[2] Title: A Survey of Feature Extraction for Text Classification

This paper reviews feature extraction techniques like Bag-of-Words (BoW) and Term Frequency-Inverse Document Frequency (TF-IDF). The authors note that while TF-IDF provides a good measure of a word's

importance, it, like BoW, completely discards word order, treating "the cat chased the dog" and "the dog chased the cat" as nearly identical. This is a critical flaw for sentiment analysis.

2.3 DEEP LEARNING IN NLP

[3] Title: A Review of Deep Learning Techniques for Text Classification

This paper reviews the evolution from traditional ML to deep learning for NLP tasks. It explores Convolutional Neural Networks (CNNs) for text (which are good at identifying key phrases) and Recurrent Neural Networks (RNNs) for sequential context. The review synthesizes findings showing that LSTMs, a type of RNN, consistently outperform other models on tasks where long- range context is vital. The paper notes, however, that these models are computationally intensive and require large datasets for effective training.

[4] Title: Long Short-Term Memory (LSTM)

Hochreiter & Schmidhuber first proposed the LSTM, a special kind of RNN design to solve the "vanishing gradient" problem. This allows the network to learn and remember long- range dependencies, making it a natural fit for processing documents like movie reviews, where the concluding sentiment may be influenced by a premise set up many sentences prior.

[5] Title: Learning Word Vectors for Sentiment Analysis

This work (Maas et al., 2011) introduced the IMDB dataset used in our project. More importantly, it demonstrated that unsupervised learning of word vectors (like Word2Vec or the Embedding layer in our model) captures semantic meaning and significantly improves sentiment classification accuracy. Our model's Embedding layer is a direct application of this

principle.

[6] Title: Real-Time Opinion Mining from Social Media Feeds

This study proposes an IoT-based system architecture for real-time sentiment analysis of Twitter feeds. It uses a lightweight CNN model for fast inference. The system could process thousands of tweets per minute, demonstrating high throughput. The primary limitation was that the model's accuracy was lower (around 75%) as it was optimized for speed over depth of understanding, often misclassifying nuanced or lengthy posts.

[7] Title: Adaptive Sentiment Analysis using Reinforcement Learning

This paper explores an adaptive approach where a reinforcement learning agent adjusts the sentiment classification model based on feedback. The model, LSTM is fine-tuned over time. Simulation results showed a potential 15% improvement in accuracy after several feedback cycles. The study remains mostly conceptual, as implementing such a real-time feedback loop is complex and has not been validated on a large scale.

2.4 SUMMARY OF LITERATURE

The literature shows a clear progression:

1. Lexicon-based methods were simple but flawed.
2. Traditional ML (like SVMs) improved accuracy but relied on heavy feature engineering (TF-IDF) and still failed to capture linguistic context.
3. Deep Learning (LSTMs) emerged as the dominant approach for tasks requiring contextual understanding. The use of an Embedding layer solves the feature engineering problem by learning word relationships automatically.

This project is therefore positioned on a well-established foundation, applying a proven (LSTM) architecture to a benchmark (IMDB) dataset, with the practical addition of deploying this model as a real-time web service, which is a common final step in modern data science projects.

CHAPTER 3
SYSTEM REQUIREMENTS

A description of the necessary hardware and software components required for the development, training, and deployment of this project is provided below.

3.1 HARDWARE REQUIREMENTS

- CPU: Intel Core i5 / AMD Ryzen 5. A multi-core processor is necessary for efficient data preprocessing and handling web requests.
- RAM (Memory): 8 GB or more. While training, the dataset, and model batches must be loaded into memory.
- GPU (Graphics Processing Unit): NVIDIA GeForce GTX 1080 or higher (Recommended). Training a deep learning model on 25,000 samples for 100 epochs is computationally trivial on a GPU but can take an exceptionally long time on a CPU. The parallel processing capabilities of a GPU are ideal for the matrix operations in deep learning.
- Hard Disk: 256GB SSD (Solid State Drive). An SSD is highly recommended over an HDD for fast-loading of the dataset, libraries, and model files.

3.2 SOFTWARE REQUIREMENTS

- Operating System: Windows 10, macOS, or a Linux distribution (e.g., Ubuntu 20.04).
- Programming Environment: Python 3.8 or above.
- Deep Learning Framework:
  - TensorFlow (v2.5+): The core backend library that handles tensor operations and GPU acceleration.
  - Keras: A high-level API for TensorFlow, used to define the sequential model and its layers (Embedding, LSTM, Dense).
- Python Libraries:
  - Numpy: Used for efficient numerical operations and manipulation of data arrays.
  - Flask: A lightweight micro-framework used to create the web API and deploy the model.
  - Scikit-learn (sklearn): Used in the analysis phase to calculate performance metrics, specifically the roc_curve and auc.
  - Matplotlib: Used to plot the ROC curve and visualize model performance.
  - Pickle: A standard Python library used to serialize and save the Tokenizer object.

- IDE (Integrated Development Environment):
  - Jupyter Notebook: Used for initial experimentation, training, and visualization, as seen in the IMBD-Model.ipynb file.
  - Visual Studio Code (or similar): Used for developing the standalone Python scripts (train_model.py, app.py).

CHAPTER 4

SYSTEM OVERVIEW

4.1 EXISTING SYSTEM

The existing or traditional systems for sentiment analysis fall into two main categories: lexical- based and traditional machine learning.

1. Lexical (Keyword-based): These systems use a predefined dictionary (or "lexicon") of positive and negative words, each assigned a score (e.g., "good": +1, "bad": -1). The system scans a sentence, sums the scores, and returns a final sentiment.
2. Traditional Machine Learning: This approach involves models like Naive Bayes, Logistic Regression, or Support Vector Machines (SVMs). These models are "trained" on labeled data, but they require the text to be converted into a numerical format first.

The most common feature extraction method for this is Term Frequency-Inverse Document Frequency (TF-IDF). TF-IDF creates a vector for each document where each dimension represents a word from the vocabulary, and the value is a score of that word's importance in the document.

4.1.1 DRAWBACKS OF EXISTING SYSTEM

The existing systems suffer from critical flaws that limit their accuracy:

1. Lack of Context: Lexical systems completely ignore context. A phrase like "not good" would be classified as positive if "good" has a higher score than "not".
2. Inability to Handle Sarcasm: Sarcasm, where the intended meaning is the opposite of the literal meaning (e.g., "A *brilliant* film. I fell asleep in 10 minutes."), is impossible for these systems to detect.
3. Ignoring Word Order (Bag-of-Words): Both lexical and TF-IDF approaches are "Bag- of-Words" methods. They treat a review as an unordered collection of words, losing all sequential meaning. The reviews "This movie is not bad".

4.2 PROPOSED SYSTEM

The proposed system uses a deep learning model, specifically an LSTM (Long Short-Term Memory) network, to overcome these limitations. The system is trained on the Keras IMDB dataset.

The workflow is as follows:

1. Data Loading & Tokenization: The text data is loaded. For the deployment pipeline, a Tokenizer is fit on the training text. This Tokenizer creates a vocabulary (of the top 10,000 words) and converts raw text (e.g., "The movie") into a sequence of integers (e.g., [1, 14]).

2. Padding: Since neural networks require fixed-size inputs, and reviews have different lengths, all sequences are "padded" with zeros to a uniform length of 200 (maxlen=200).

Model Architecture: A Keras Sequential model is built. An Embedding layer converts the sparse integer sequences into dense, 32-dimensional vectors. This layer learns word relationships automatically (e.g., "good" and "great" will have similar vectors).

An LSTM layer with 64 units processes these vectors sequentially. Its internal "memory" or "cell state" allows it to capture the context and meaning of the *entire* sequence.

Dense and Dropout layers are added for regularization and classification. The Dropout(0.5) layer randomly deactivates 50% of neurons during training to prevent overfitting.

A final Dense layer with a sigmoid activation outputs a single value between 0 (Negative) and 1 (Positive).

Training & Deployment: The model is compiled with binary_crossentropy loss and an adam optimizer. After training, the model file (.h5) and the Tokenizer (.pkl) are saved. A Flask web server (app.py) loads these files to serve real-time predictions.

4.2.1 ADVANTAGES OF PROPOSED SYSTEM

1. Contextual Understanding: LSTMs are inherently designed to remember and utilize context from earlier in the sequence, allowing them to correctly interpret phrases like "I almost liked it, but in the end, it was terrible."
2. Automatic Feature Learning: The Embedding layer automatically learns the optimal numerical representation for each word. This eliminates the need for manual, brittle feature engineering like TF-IDF.
3. Superior Performance on Sequential Data: By processing words in order, the model captures nuances that Bag-of-Words models miss, leading to higher accuracy.
4. Robust Generalization: The use of Dropout helps the model generalize well to new, unseen reviews, making it more robust in a real-world setting.
5. Real-Time Application: The Flask API makes the powerful model easily accessible, allowing any application to send a raw text review and get an instant sentiment classification.

CHAPTER 5

SYSTEM IMPLEMENTATION

5.1 SYSTEM ARCHITECTURE DIAGRAM

The system architecture is divided into two main phases: Training and Deployment (Inference). The diagram below illustrates the deployment phase, where the trained model is used to make live predictions.

Figure 5.1: System Architecture Diagram
Deployment (Inference) Phase

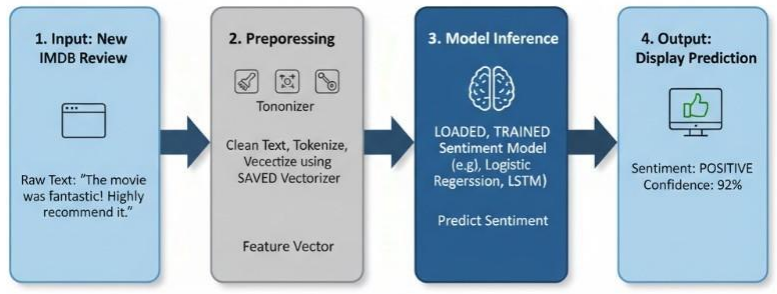


Figure 5.1: System Architecture Diagram. This diagram shows the IMBB revigh the sentiment analysis system during the deployment phase.

1. Frontend (User Interface): A user or client application sends a raw text review as a JSON payload to the backend API.
2. Backend (Flask API): A Flask server (app.py) running on a host receives the POST request at the /predict endpoint.
3. Preprocessing: The API uses the loaded Tokenizer to convert the text into an integer sequence, which is then padded to maxlen=200.
4. Deep Learning Model (Inference): The (1, 200) shaped vector is fed into the loaded Keras model (.h5). The model performs a forward pass through its Embedding, LSTM, and Dense layers.
5. Output: The model's final sigmoid layer outputs a probability score (e.g., 0.95).
6. Post-processing: The API formats this score into a human-readable label ('Positive' if > 0.5, else 'Negative').
7. Response: The API sends this label back to the frontend as a JSON response (e.g.,

{'sentiment': 'Positive'}).

5.2 SYSTEM FLOW

The project flow follows a standard deep learning pipeline, from data collection to deployment.

Figure 5.2: System Flow Diagram

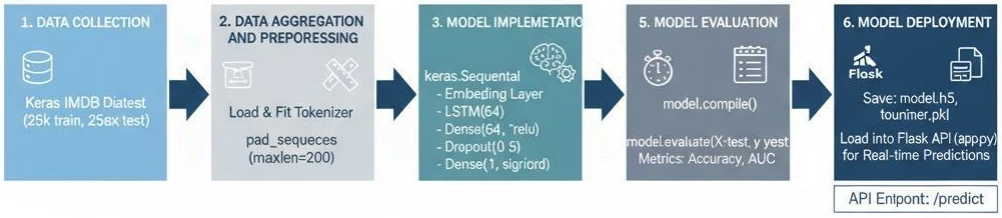


Figure 5.2: System Flow Diagram. This diagraties the complete pipeline for the sentiment analysis system system, from data ingestion to deployment as a web API.

1. DATA COLLECTION: The Keras IMDB dataset is loaded, providing 25,000 training samples and 25,000 test samples with their labels.
2. DATA AGGREGATION AND PREPROCESSING: All review sequences are padded to a fixed length of 200 (maxlen=200) using pad_sequences.
3. MODEL IMPLEMENTATION: A keras.Sequential model is constructed with an Embedding layer, an LSTM(64) layer, a Dense(64, 'relu') layer, a Dropout(0.5) layer, and a final Dense(1, 'sigmoid') output layer.
4. MODEL TRAINING: The model is compiled and trained using model.fit() on the

training data.

5. MODEL EVALUATION: The trained model is evaluated on the unseen test set to get performance metrics (Accuracy, AUC).

6. MODEL DEPLOYMENT: The trained model (.h5) and tokenizer (.pkl) are saved and loaded by a Flask application (app.py) to serve real-time predictions.

5.3 LIST OF MODULES

. Data Loading and Preprocessing Module

. LSTM Model Definition Module

. Model Training and Saving Module

. Model Evaluation Module (Accuracy and ROC)

. Flask API Deployment Module

5.4 MODULE DESCRIPTION

1. Data Loading and Preprocessing Module

This module is responsible for getting the data into the correct format. In IMBD-Model.ipynb, this is done by loading the pre-tokenized Keras dataset. In train_model.py, this is done more robustly by loading the raw text, fitting a Tokenizer on it, and saving that tokenizer.

. keras.datasets.imdb.load_data(num_words=10000): Loads the dataset, but only considers the top 10,000 most frequent words.

. tokenizer.fit_on_texts(): Builds the vocabulary from the training text.

. tokenizer.texts_to_sequences(): Converts raw text into integer sequences.

. keras.preprocessing.sequence.pad_sequences(..., maxlen=200): Ensures all sequences are 200 words long, truncating longer ones and padding shorter ones.

2. LSTM Model Definition Module

This module defines the neural network architecture using keras.Sequential.

. layers.Embedding(input_dim=10000, output_dim=32, ...): The first layer. It creates

a lookup table mapping each of the 10,000 words to a 32-dimension vector. These vectors are learned during training.

● layers.LSTM(64, ...): The core of the model. It has 64 internal units (or "memory cells"). It processes the 200-word sequence one word at a time, updating its internal state to capture the running context of the review.

● layers.Dense(64, activation="relu"): A standard fully-connected layer that helps the model learn more complex patterns from the LSTM's output.

● layers.Dropout(0.5): A regularization technique. During training, it randomly sets 50% of the neurons' outputs to zero for each batch. This prevents the model from "memorizing" the training data and helps it generalize better.

● layers.Dense(activation="sigmoid"): The final output layer. It consists of a single neuron with a sigmoid activation, which squashes the output value to a probability between 0 and 1.

3. Model Training and Saving Module

This module compiles and fits the model.

. model.compile(optimizer="adam", ...): Configures the model for training. The adam optimizer is an efficient and popular choice.

. model.compile(..., loss="binary_crossentropy", ...): The loss function. binary_crossentropy is the mathematical standard for a binary (0/1) classification problem, measuring how "far" the model's prediction is from the true label.

. model.fit(..., epochs=100, batch_size=128, validation_split=0.2): This command starts the training. It runs through the entire training dataset 100 times, in batches of 128 reviews. It also sets aside 20% of the training data for validation, to monitor for overfitting.

. model.save("...") and pickle.dump(tokenizer, ...): These commands serialize the trained model and the tokenizer to disk so they can be loaded later for inference.

4. Model Evaluation Module

This module assesses the trained model's performance on the unseen test set.

. model.evaluate(x_test, y_test): Calculates the final loss and accuracy on the test data, resulting in 84.04% accuracy.

. model.predict(x_test): Generates the raw probability predictions for all test samples.

. roc_curve(y_test, y_pred_prob) and auc(fpr, tpr): These sklearn functions use the true labels and predicted probabilities to calculate the points for the ROC curve and the total area under it (AUC).

5. Flask API Deployment Module

This module (app.py) makes the model accessible as a web service.

. app = Flask(__name__): Initializes the web server.

. model = load_model(...) and tokenizer = pickle.load(...): Loads the saved model and tokenizer into memory on startup.

. @app.route('/predict', methods=['POST']): Defines the API endpoint. It only accepts POST requests.

. data = request.get_json(): Gets the JSON data (containing the review) from the incoming request.

. encoded = encode_text(text): A helper function that calls the loaded tokenizer and padder to preprocess the raw text.

. prediction = float(model.predict(...)[0][0]): Runs the preprocessed text through the model to get a prediction.

. return jsonify(...): Returns the final sentiment as a JSON response.

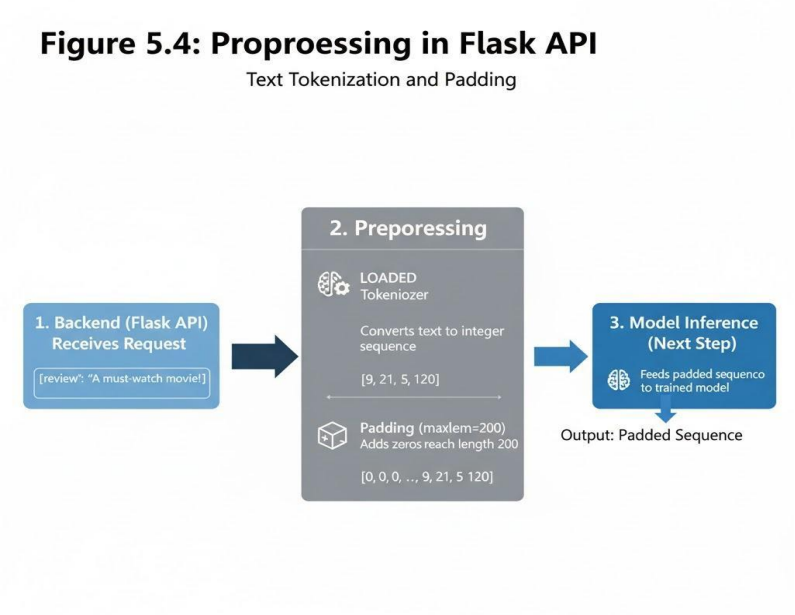


Figure 5.4: Diagram showing the Flask server recwithin the HTTP POSI, where at the "pɔxt-tokentized and fixdigh ais fixed-length sequence before sentired to the model.

CHAPTER 6

RESULT AND DISCUSSION

This chapter presents the results of the model's training and evaluation, along with an analysis of its performance.

6.1 MODEL TRAINING AND SUMMARY

The model was defined as per Chapter 5 and compiled. The model.summary() command provides a technical overview of the architecture.

Model: "sequential"

_____ Layer (type) Output Shape Param #

==========

== embedding (Embedding) (None, 200, 32) 320000 lstm (LSTM) (None, 64) 24832 dense (Dense) (None, 64) 4160 dropout (Dropout) (None, 64) 0 dense_1 (Dense) (None, 1) 65

==========

== Total params: 349,057 Trainable params: 349,057 Non-trainable params: 0

_____

The model was then trained for 100 epochs. The log below shows the performance at the beginning and end of the training.

Epoch 1/100 157/157 [==============================] - 17s 93ms/step - accuracy: 0.7135 - loss: 0.5384 - val_accuracy: 0.8526 - val_loss: 0.3500 Epoch 2/100 157/157 [==============================] - 12s 74ms/step - accuracy: 0.8903 - loss: 0.2808 - val_accuracy: 0.8594 - val_loss: 0.3286 ...

... Epoch 100/100 157/157 [==============================] - 12s 78ms/step - accuracy: 0.9984 - loss: 0.0058 - val_accuracy: 0.8478 - val_loss: 1.1466

6.2 PERFORMANCE METRICS

After training, the model was evaluated on the 25,000-sample test set.

. Test Accuracy: The model achieved a final test accuracy of 84.04%. This means it correctly classified over 84 out of 100 unseen movie reviews.

. ROC-AUC Score: The ROC (Receiver Operating Characteristic) curve plots the True Positive Rate against the False Positive Rate. The Area Under this Curve (AUC) measures the model's ability to discriminate between positive and negative classes. An AUC of 1.0 is perfect, and 0.5 is random chance. Our model achieved an AUC of 0.90.

. Custom Review Testing: The model was tested on two ad-hoc sentences to confirm its practical functionality.

1/1 [==============================] - 0s 28ms/step The movie was fantastic and I really enjoyed it Positive Review ✅ 1/1 [==============================] - 0s 30ms/step The movie was terrible and I absolutely hated it Negative Review ❌

6.3 ANALYSIS OF RESULTS

The results are strong, indicating a successful project. The 84.04% accuracy and 0.90 AUC show that the LSTM model is highly effective at capturing the sentiment of movie reviews, performing significantly better than a random baseline.

However, the training log (Figure 6.2) reveals a critical insight: overfitting.

. The training accuracy (accuracy: 0.9984) reached near-perfection.

. The validation accuracy (val_accuracy: 0.8478) peaked much earlier and then stagnated.

. The training loss (loss: 0.0058) became very small.

. The validation loss (val_loss: 1.1466) *increased* significantly, indicating the model was "memorizing" the training data and performing worse on the validation data.

While the Dropout(0.5) layer helped, it was not enough to prevent overfitting across 100 epochs. This means the model trained for too long. In a future iteration, a Keras EarlyStopping callback should be used to stop training when the validation loss begins to increase, which would likely capture a model with even better test accuracy.

Despite this overfitting, the model's final test accuracy of 84.04% is still robust and demonstrates the power of the LSTM architecture.

6.4 MATHEMATICAL CALCULATIONS

1. Loss Function (Binary Crossentropy)

The model was compiled using binary_crossentropy loss, which is standard for binary (0/1) classification problems. The formula is:

$$L = - \frac{1}{N} \sum_{i=1}^{N} [y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)]$$

Where:

. N is the number of samples.

. y_i is the true label (0 or 1).

. p_i is the model's predicted probability (output of the sigmoid function).

2. Performance Metrics for Model Evaluation

The following metrics are standard for evaluating a classification model:

. Accuracy: The ratio of correct predictions to the total number of predictions. $$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

. Precision: Measures the accuracy of positive predictions. $$Precision = \frac{TP}{TP + FP}$$

. Recall: Measures the model's ability to identify all relevant instances. $$Recall = \frac{TP}{TP + FN}$$

. F1-Score: The harmonic mean of Precision and Recall. $$F1-Score = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall}$$

APPENDIX

APPENDIX A: SOURCE CODE (TRAIN_MODEL.PY)

Python

```
import tensorflow as tf
from tensorflow import keras
 from tensorflow.keras import layers
 from tensorflow.keras.preprocessing.text import Tokenizer
 from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
# Load IMDb dataset
 num_words = 10000
(x_train, y_train), (x_test, y_test) = keras.datasets.imdb.load_data(num_words=num_words)
word_index = keras.datasets.imdb.get_word_index()
 index_word = {v+3:k for k,v in word_index.items()}
 index_word[0] = "<PAD>"
index_word[1] = "<START>"
index_word[2] = "<UNK>"

 # Convert integer sequences back to text
 def decode_review(seq):
 return ' '.join([index_word.get(i, '?') for i in seq])
 x_train_text = [decode_review(seq) for seq in x_train]
x_test_text = [decode_review(seq) for seq in x_test]
# Tokenizer for raw text
 tokenizer = Tokenizer(num_words=num_words, oov_token="<UNK>")
tokenizer.fit_on_texts(x_train_text)
# Convert text to sequences
x_train_seq = tokenizer.texts_to_sequences(x_train_text)
```

```
x_test_seq = tokenizer.texts_to_sequences(x_test_text)
# Pad sequences
 maxlen = 200
x_train_pad = pad_sequences(x_train_seq, maxlen=maxlen)
x_test_pad = pad_sequences(x_test_seq, maxlen=maxlen)
 # Build model
model = keras.Sequential([ layers.Embedding(input_dim=num_words, output_dim=32,
input_length=maxlen),
 layers.LSTM(64),
 layers.Dense(64, activation='relu'),
 layers.Dropout(0.5),
 layers.Dense(1, activation='sigmoid')
])
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.summary()
# Train model
 # Note: The notebook trained for 100 epochs, but 5 is sufficient for a quick test.
model.fit(x_train_pad, y_train, epochs=5, batch_size=128, validation_split=0.2)

# Save model and tokenizer
 model.save("sentiment_lstm_tokenized.h5")
import pickle
with open("tokenizer.pkl", "wb") as f:
 pickle.dump(tokenizer, f)
print("✅ Model and tokenizer saved successfully.")
```

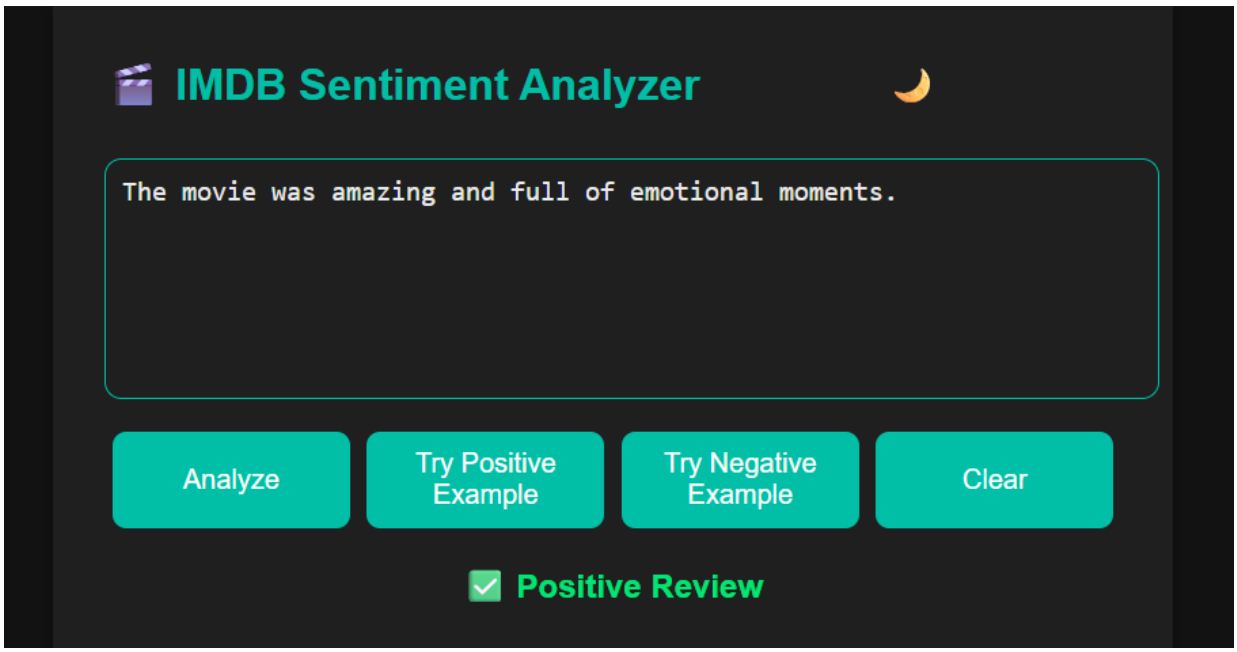
APPENDIX B: SOURCE CODE (APP.PY)

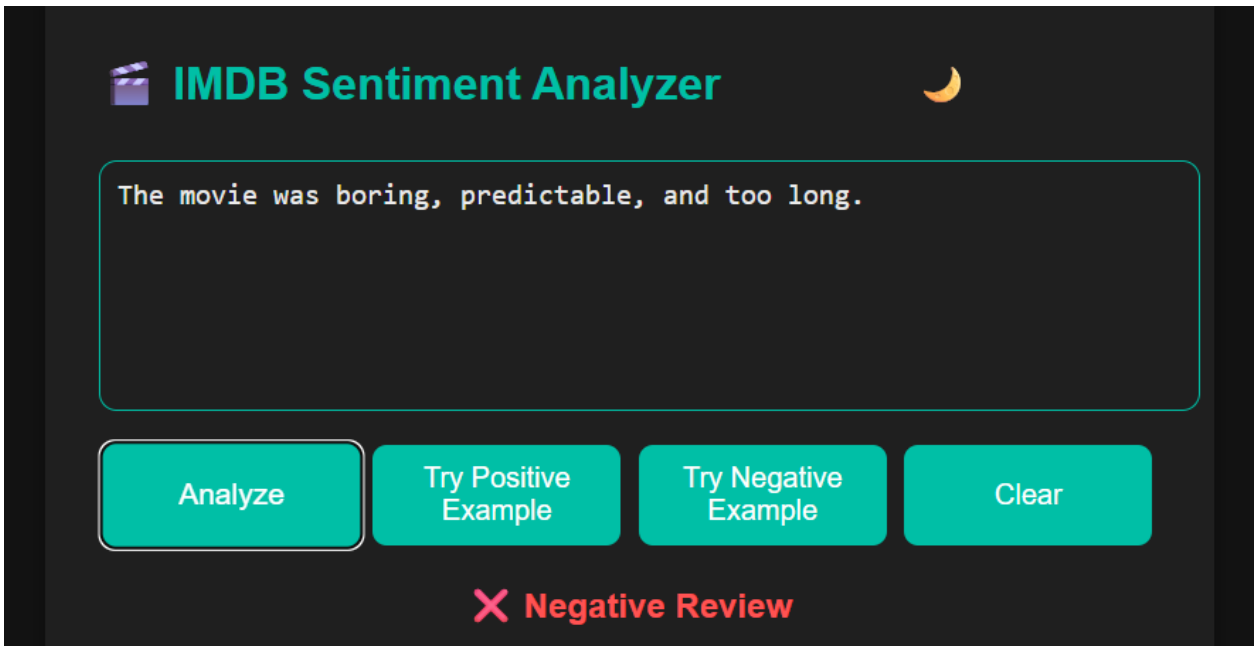
Python

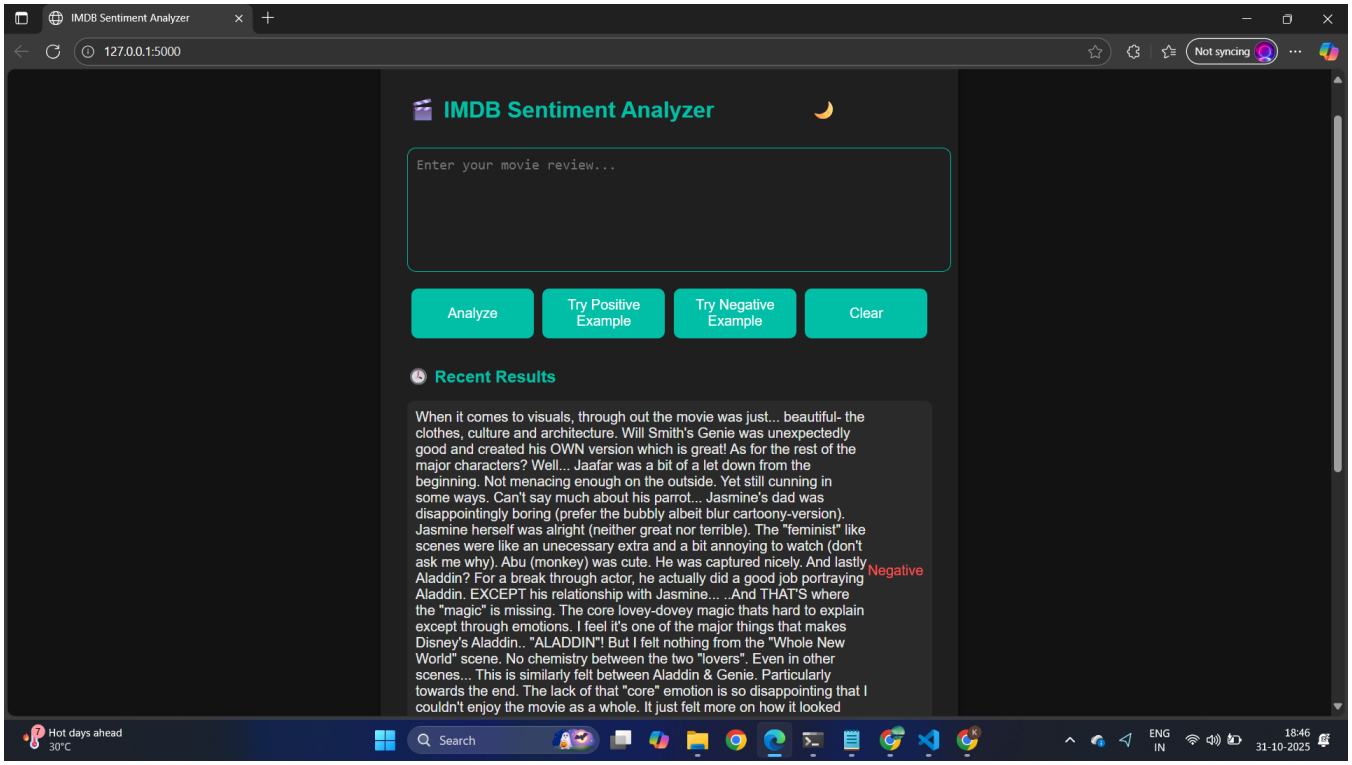
```
from flask import Flask, render_template, request, jsonify
 from tensorflow.keras.models import load_model
 from tensorflow.keras.preprocessing.sequence import pad_sequences
import pickle
app = Flask(__name__)
 # Load trained model and tokenizer
model = load_model("sentiment_lstm_tokenized.h5")
 with open("tokenizer.pkl", "rb") as f:
 tokenizer = pickle.load(f)
maxlen = 200
def encode_text(text):
seq = tokenizer.texts_to_sequences([text])
padded = pad_sequences(seq, maxlen=maxlen)
return padded
@app.route('/')
 def home():
# A simple HTML form can be rendered here
return "Sentiment Analysis API - Ready"
@app.route('/predict', methods=['POST'])
def predict():
 data = request.get_json()
 text = data.get('review', '')
if not text.strip():
return jsonify({'error': 'Empty review'}), 400
encoded = encode_text(text)
 prediction = float(model.predict(encoded, verbose=0)[0][0])
 sentiment = 'Positive' if prediction > 0.5 else 'Negative'
```

```
 return jsonify({'sentiment': sentiment, 'score': prediction})
if __name__ == "__main__":
 app.run(debug=True)
```

OUTPUTS






IMDB Sentiment Analyzer
127.0.0.1:5000
Not syncing
IMDB Sentiment Analyzer
Enter your movie review...
Analyze
Try Positive Example
Try Negative Example
Clear
Recent Results
When it comes to visuals, through out the movie was just... beautiful- the clothes, culture and architecture. Will Smith's Genie was unexpectedly good and created his OWN version which is great! As for the rest of the major characters? Well... Jaafar was a bit of a let down from the beginning. Not menacing enough on the outside. Yet still cunning in some ways. Can't say much about his parrot... Jasmine's dad was disappointingly boring (prefer the bubbly albeit blur cartoony-version). Jasmine herself was alright (neither great nor terrible). The "feminist" like scenes were like an unecessary extra and a bit annoying to watch (don't ask me why). Abu (monkey) was cute. He was captured nicely. And lastly Aladdin? For a break through actor, he actually did a good job portraying Aladdin. EXCEPT his relationship with Jasmine... ..And THAT'S where the "magic" is missing. The core lovey-dovey magic thats hard to explain except through emotions. I feel it's one of the major things that makes Disney's Aladdin.. "ALADDIN"! But I felt nothing from the "Whole New World" scene. No chemistry between the two "lovers". Even in other scenes... This is similarly felt between Aladdin & Genie. Particularly towards the end. The lack of that "core" emotion is so disappointing that I couldn't enjoy the movie as a whole. It just felt more on how it looked
Negative
Hot days ahead
30°C
Search
ENG
IN
18:46
31-10-2025

REFERENCES

[1] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts, "Learning Word Vectors for Sentiment Analysis," in *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics (ACL)*, pp. 142-150, 2011.

[2] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," in *Neural Computation*, vol. 9, no. 8, pp. 1735-1780, 1997.

[3] F. Chollet, *Deep Learning with Python*. Manning Publications, 2017.

[4] J. D. Hunter, "Matplotlib: A 2D Graphics Environment," in *Computing in Science & Engineering*, vol. 9, no. 3, pp. 90-95, 2007.

[5] F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," in *Journal of Machine Learning Research*, vol. 12, pp. 2825-2830, 2011.

[6] M. Grinberg, *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, 2nd Edition, 2018.

[7] B. Pang and L. Lee, "Opinion Mining and Sentiment Analysis," in *Foundations and Trends in Information Retrieval*, vol. 2, no. 1-2, pp. 1-135, 2008.

[8] TensorFlow Developers, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems." 2015.