Deep Learning and Reinforcement Learning

Project Title: Character-Level Text Generation

Students: Poojitha B - 1BG23CS100
Rachana F Patil - 1BG23CS113

Project Objective

The project aims to build a text generation system that can produce character- or word-level continuations based on classical literature styles, using LSTM models and an interactive Streamlit interface.

Key Objectives:

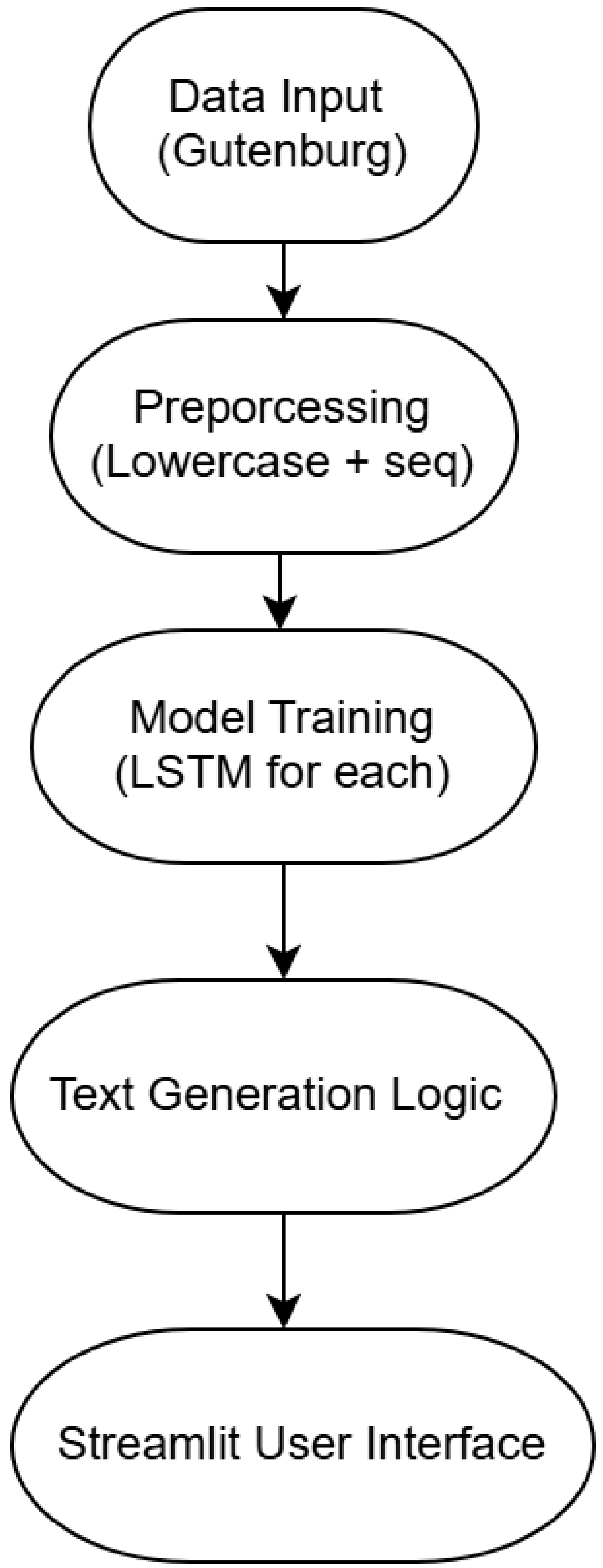
- Train character-level LSTM models on classic books from the Gutenberg corpus.
- Generate text continuations from user prompts.
- Provide both model-generated and verbatim (book-extracted) outputs.
- Build a user-friendly app using Streamlit for real-time interaction.
- Allow style selection across multiple authors and genres.

Methodology

1. Data Collection
 – Load 5 classical texts from the NLTK Gutenberg corpus
 – Convert all text to lowercase
2. Preprocessing
 – Generate character sequences (length = 40, step = 3)
 – Tokenize characters and assign indices
 – Vectorize input and output using one-hot encoding
3. Model Design & Training
 – Build LSTM model (128 units) with softmax Dense layer
 – Compile using categorical crossentropy and Adam optimizer
 – Train one model per book (3 epochs each)

4. Text Generation Logic
 – Predict next characters using trained LSTM
 – For comparison, extract text directly from books (verbatim)
 – Support both character-wise and word-wise modes
5. Streamlit App Development
 – Interactive UI with dropdowns and input fields
 – Users select book, input prompt, and choose output type
 – Display generated or extracted text in real-time

Workflow



Key Assumptions

- The user-entered prompt exists in the original book text.
- Each book is trained individually; no transfer learning between styles.
- Character-level generation is sufficient to capture the book's writing style.
- Only lowercase text is used to simplify the model vocabulary.
- Model output is evaluated qualitatively (no automatic accuracy metrics).
- Verbatim output functions rely on simple string search and slicing.
- The model's training is limited to 3 epochs, prioritizing speed over perfection.
- The Streamlit app assumes clean and correct user input.

Model Evaluation and Analysis

1. Training Observations
 – Models trained for 3 epochs using LSTM and categorical crossentropy
 – Loss decreased steadily, indicating learning
2. Output Quality
 – Generated text mimics original tone and vocabulary
 – Some outputs may be repetitive or less coherent
3. Strengths
 – Captures basic literary style
 – Verbatim mode gives clean and accurate references
4. Limitations
 – No quantitative metrics used
 – Short training and character-level modeling limit depth

Project Summary and Outcomes

This project successfully implemented a character-level text generator using LSTM models trained on classic literature. A Streamlit app was developed for interactive input and output, offering both generated and verbatim text extraction.

Key Outcomes:

- Built and trained individual LSTM models for five classic books
- Enabled user-driven prompt-based text generation (character/word)
- Created a user-friendly Streamlit interface for real-time interaction
- Demonstrated the stylistic influence of training data in generated text

Future Improvements and Extensions

- Increase Training Epochs
  Improve model learning and reduce repetition
- Add Evaluation Metrics
  Use Perplexity or BLEU Score to measure text quality
- Integrate Style Transfer
  Allow mixing or switching between author styles
- Enhance UI Features
  Show model confidence and compare generated outputs
- Add Autocorrect for Prompts
  Fix typos automatically to prevent “Prompt not found” issues

Reflections and Learning Outcomes

1. **LSTM (Long Short-Term Memory)**
 Understood how LSTM networks retain and use past information for sequence prediction.
2. **One-Hot Encoding**
 Learned how text is converted into numerical format for training neural networks.
3. **Sequential Model Building**
 Gained hands-on experience in building and compiling models using Keras Sequential API.
4. **Text Preprocessing**
 Explored techniques like lowercasing, character indexing, and vectorization for preparing raw data.
5. **Interactive Deployment (Streamlit)**
 Learned how to deploy models with a clean user interface for real-time input/output.

Appendix

• Code Files

– logic.ipynb : Contains data preprocessing, LSTM model building, training, and text generation logic

– app.py : Streamlit-based web interface for user interaction with the trained models

• Dataset

– NLTK's Gutenberg corpus (e.g., austen-emma.txt, bible-kjv.txt)

• Libraries & Tools Used

– TensorFlow, Keras, NumPy, NLTK, Streamlit

Thank you!