Project Documentation: Text Generation with Markov Chains

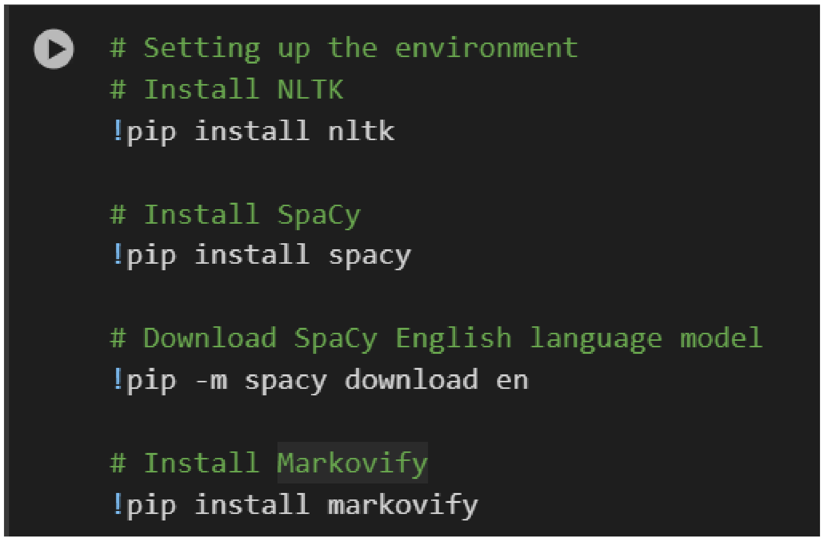
Overview

In this task we have to implement a simple text generation algorithm using Markov Chains. This task involves creating a statistical model that predicts the probability of a character or word based on previous one.

Setup and Installation

1. Setting up the Environment:

To start our project on text generation using Markov chains, we need to install and set up several libraries. The following code does just that:

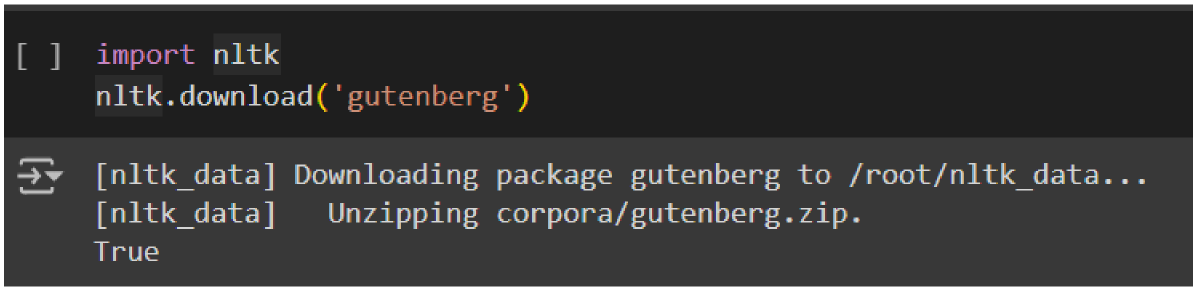


```
# Setting up the environment
# Install NLTK
!pip install nltk

# Install SpaCy
!pip install spacy

# Download SpaCy English language model
!pip -m spacy download en

# Install Markovify
!pip install markovify
```

1. **NLTK (Natural Language Toolkit):** This library is essential for various text processing and natural language processing tasks.
2. **SpaCy:** A popular library for advanced natural language processing, SpaCy provides tools for tokenizing and processing text.
3. **SpaCy English Language Model:** Downloading this model allows SpaCy to process English text effectively.
4. **Markovify:** This library is used for creating Markov chain-based text generators, which is the core technique for our text generation task.

2. Downloading the Gutenberg Corpus:

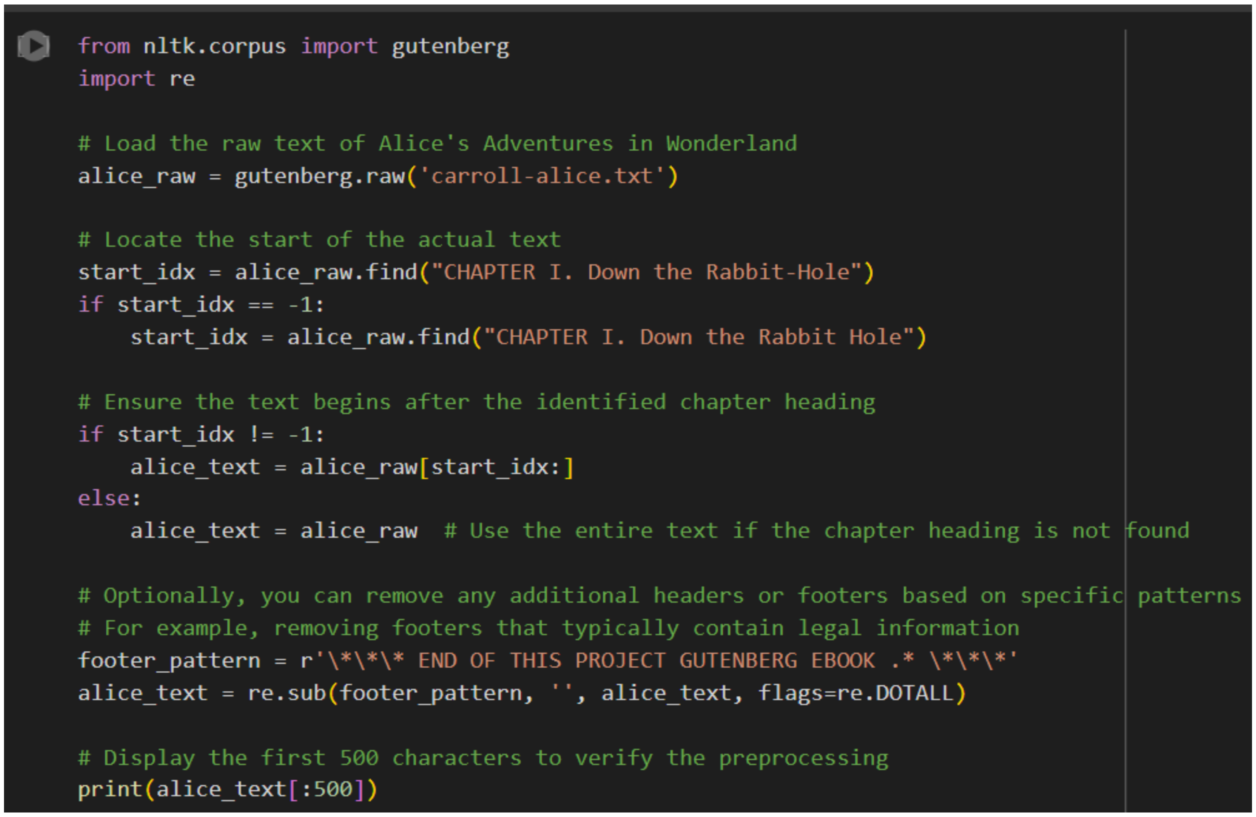
The following code snippet uses the NLTK library to download the Gutenberg corpus:



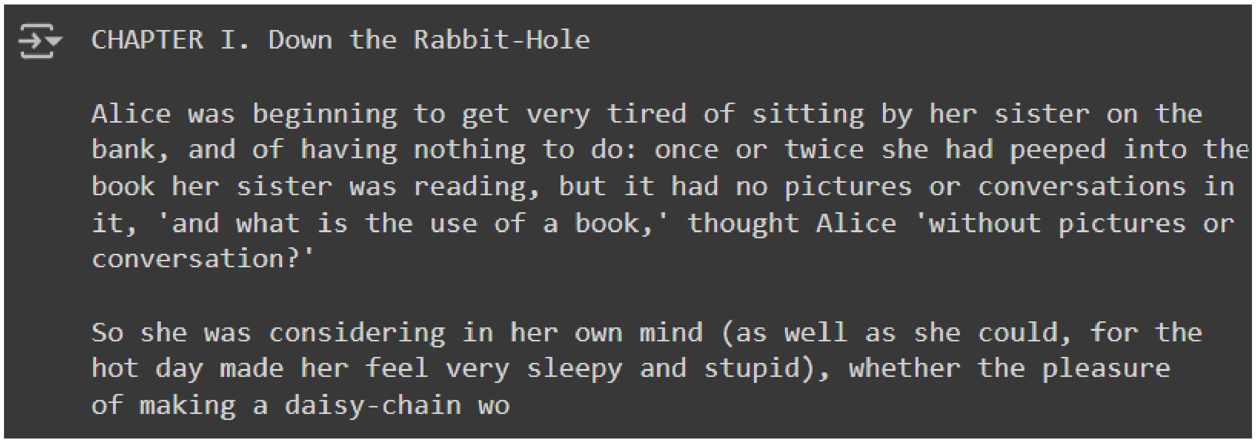
```
[ ]  import nltk
     nltk.download('gutenberg')

     [nltk_data] Downloading package gutenberg to /root/nltk_data...
     [nltk_data]   Unzipping corpora/gutenberg.zip.
     True
```

- **import nltk:** This line imports the NLTK library, which provides tools for text processing and natural language processing tasks.
- **nltk.download('gutenberg'):** This command downloads the Gutenberg corpus, a collection of classic literary works that is included in the NLTK library. This corpus can be used for various text analysis and natural language processing tasks.

3. The provided code snippet loads and preprocesses the text of "Alice's Adventures in Wonderland" from the NLTK Gutenberg corpus. It involves these steps:



```
from nltk.corpus import gutenberg
import re

# Load the raw text of Alice's Adventures in Wonderland
alice_raw = gutenberg.raw('carroll-alice.txt')

# Locate the start of the actual text
start_idx = alice_raw.find("CHAPTER I. Down the Rabbit-Hole")
if start_idx == -1:
    start_idx = alice_raw.find("CHAPTER I. Down the Rabbit Hole")

# Ensure the text begins after the identified chapter heading
if start_idx != -1:
    alice_text = alice_raw[start_idx:]
else:
    alice_text = alice_raw  # Use the entire text if the chapter heading is not found

# Optionally, you can remove any additional headers or footers based on specific patterns
# For example, removing footers that typically contain legal information
footer_pattern = r'\*\*\* END OF THIS PROJECT GUTENBERG EBOOK .* \*\*\*'
alice_text = re.sub(footer_pattern, '', alice_text, flags=re.DOTALL)

# Display the first 500 characters to verify the preprocessing
print(alice_text[:500])
```



```
CHAPTER I. Down the Rabbit-Hole

Alice was beginning to get very tired of sitting by her sister on the
bank, and of having nothing to do: once or twice she had peeped into the
book her sister was reading, but it had no pictures or conversations in
it, 'and what is the use of a book,' thought Alice 'without pictures or
conversation?'

So she was considering in her own mind (as well as she could, for the
hot day made her feel very sleepy and stupid), whether the pleasure
of making a daisy-chain wo
```

1. **Loading the Text**: The raw text of the book is loaded using `gutenberg.raw('carroll-alice.txt')`.
2. **Finding the Start of the Story**: The code searches for the start of the actual story by locating the chapter heading "CHAPTER I. Down the Rabbit-Hole". If not found, it looks for "CHAPTER I. Down the Rabbit Hole".
3. **Extracting the Main Text**: Once the chapter heading is found, the text is sliced to begin from that point. If not found, the entire text is used.
4. **Removing Footers**: Optionally, the code removes any footer text containing legal information using a regular expression.
5. **Verification**: The first 500 characters of the preprocessed text are printed to verify the steps.

Conclusion:

In this project, I successfully implemented a trigram-based Markov chain model for text generation using the "Alice's Adventures in Wonderland" dataset. By leveraging Python, NLTK, SpaCy, and Markovify, I was able to preprocess the text and build a model that generates coherent and contextually relevant text. This project not only demonstrated my proficiency in natural language processing and text generation techniques but also highlighted the potential of Markov models in generating meaningful text. The skills and insights gained from this project are valuable for future applications in NLP and AI-driven text generation tasks.