**Assignment 4). AutoComplete**

ChatGPT: https://chat.openai.com/share/04673f7c-d66a-41d2-b408-c999f804f319

\_\_\_\_\_\_\_\_

Human: The 1980s saw a shift from Natural Language Processing techniques aiming to codify the grammatical rules of natural language towards techniques aiming to use statistical models to generate text. One early idea which technically isn’t “AI” seeing as it is “memorizing” the training data and yet introduces us to the power contained in statistical techniques of text generation is the idea of Markov chains. Write a python function generate(filename: str, start\_words: list[str], chain\_length: int, num\_generated: int) -> str which takes a filename, a chain length, a list of start words which has to be exactly as long as the chain\_length (why?), and an integer num\_generated and returns a sentence num\_generated words long which sounds similar to the text contained in filename.

ChatGPT: To create a Python function that generates a sentence using Markov chains based on a given text file, you can follow the steps below:Read the text from the file.Tokenize the text into words.Create a Markov chain dictionary based on the chain length.Generate a sentence using the Markov chain.Here's the implementation:

pythonCopy code

import random

def read\_text(filename):

with open(filename, 'r', encoding='utf-8') as file:

return file.read()

def tokenize\_text(text):

return text.split()

def create\_markov\_chain(tokens, chain\_length):

chain = {}

for i in range(len(tokens) - chain\_length):

prefix = tuple(tokens[i:i + chain\_length])

suffix = tokens[i + chain\_length]

if prefix in chain:

chain[prefix].append(suffix)

else:

chain[prefix] = [suffix]

return chain

def generate\_sentence(chain, start\_words, num\_generated):

sentence = list(start\_words)

current\_prefix = tuple(start\_words)

for \_ in range(num\_generated):

if current\_prefix not in chain:

break

next\_word = random.choice(chain[current\_prefix])

sentence.append(next\_word)

current\_prefix = tuple(sentence[-len(start\_words):])

return ' '.join(sentence)

def generate(filename, start\_words, chain\_length, num\_generated):

text = read\_text(filename)

tokens = tokenize\_text(text)

chain = create\_markov\_chain(tokens, chain\_length)

return generate\_sentence(chain, start\_words, num\_generated)

# Example usage:

filename = 'your\_text\_file.txt'

start\_words = ['The', 'quick']

chain\_length = 2 # Adjust the chain length as needed

num\_generated = 10 # Adjust the number of words to generate as needed

generated\_sentence = generate(filename, start\_words, chain\_length, num\_generated)

print(generated\_sentence)

Make sure to replace 'your\_text\_file.txt' with the path to your text file. Adjust chain\_length and num\_generated according to your preferences and requirements.

—

Human: We do not have a text file so create a random file lorem ipsum

ChatGPT: To create a random Lorem Ipsum text file for testing purposes, you can generate Lorem Ipsum text using Python's faker library and then save it to a file. If you haven't installed the faker library, you can install it using pip:

Copy code

pip install faker

Here's how you can generate a Lorem Ipsum text file:

pythonCopy code

from faker import Faker

def generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=5, sentences\_per\_paragraph=5):

fake = Faker()

fake.seed(42) # Setting seed for reproducibility

lorem\_ipsum\_text = ""

for \_ in range(num\_paragraphs):

paragraph = fake.paragraphs(nb=sentences\_per\_paragraph)

lorem\_ipsum\_text += "\n\n".join(paragraph) + "\n\n"

return lorem\_ipsum\_text

def save\_text\_to\_file(text, filename):

with open(filename, "w", encoding="utf-8") as file:

file.write(text)

# Generate Lorem Ipsum text

lorem\_ipsum\_text = generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=10, sentences\_per\_paragraph=5)

# Save the text to a file

filename = "LoremIpsum.txt"

save\_text\_to\_file(lorem\_ipsum\_text, filename)

print(f"Lorem Ipsum text saved to '{filename}'")

This code will generate a file named "LoremIpsum.txt" containing Lorem Ipsum text with 10 paragraphs, each having 5 sentences. You can adjust the num\_paragraphs and sentences\_per\_paragraph parameters according to your requirements. Once the file is generated, you can use it with the previously provided generate() function to generate text using Markov chains.

—

Human: Merge the complete code

ChatGPT: Here's the complete code merging both the Lorem Ipsum text generation and the Markov chain text generation:

pythonCopy code

import random

from faker import Faker

def generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=5, sentences\_per\_paragraph=5):

fake = Faker()

fake.seed(42) # Setting seed for reproducibility

lorem\_ipsum\_text = ""

for \_ in range(num\_paragraphs):

paragraph = fake.paragraphs(nb=sentences\_per\_paragraph)

lorem\_ipsum\_text += "\n\n".join(paragraph) + "\n\n"

return lorem\_ipsum\_text

def save\_text\_to\_file(text, filename):

with open(filename, "w", encoding="utf-8") as file:

file.write(text)

def read\_text(filename):

with open(filename, 'r', encoding='utf-8') as file:

return file.read()

def tokenize\_text(text):

return text.split()

def create\_markov\_chain(tokens, chain\_length):

chain = {}

for i in range(len(tokens) - chain\_length):

prefix = tuple(tokens[i:i + chain\_length])

suffix = tokens[i + chain\_length]

if prefix in chain:

chain[prefix].append(suffix)

else:

chain[prefix] = [suffix]

return chain

def generate\_sentence(chain, start\_words, num\_generated):

sentence = list(start\_words)

current\_prefix = tuple(start\_words)

for \_ in range(num\_generated):

if current\_prefix not in chain:

break

next\_word = random.choice(chain[current\_prefix])

sentence.append(next\_word)

current\_prefix = tuple(sentence[-len(start\_words):])

return ' '.join(sentence)

def generate(filename, start\_words, chain\_length, num\_generated):

text = read\_text(filename)

tokens = tokenize\_text(text)

chain = create\_markov\_chain(tokens, chain\_length)

return generate\_sentence(chain, start\_words, num\_generated)

# Generate Lorem Ipsum text

lorem\_ipsum\_text = generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=10, sentences\_per\_paragraph=5)

# Save the text to a file

filename = "LoremIpsum.txt"

save\_text\_to\_file(lorem\_ipsum\_text, filename)

print(f"Lorem Ipsum text saved to '{filename}'")

# Example usage:

start\_words = ['Lorem', 'ipsum']

chain\_length = 2 # Adjust the chain length as needed

num\_generated = 50 # Adjust the number of words to generate as needed

generated\_sentence = generate(filename, start\_words, chain\_length, num\_generated)

print(generated\_sentence)

This code will generate a file named "LoremIpsum.txt" containing Lorem Ipsum text and then generate a sentence using Markov chains based on that text. Adjust the start\_words, chain\_length, and num\_generated variables according to your preferences and requirements.

—

Human: Gives error TypeError Traceback (most recent call last)

<ipython-input-4-0078e642fb74> in <cell line: 55>()

53

54 # Generate Lorem Ipsum text

---> 55 lorem\_ipsum\_text = generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=10, sentences\_per\_paragraph=5)

56

57 # Save the text to a file

1 frames

/usr/local/lib/python3.10/dist-packages/faker/proxy.py in \_\_getattribute\_\_(self, attr)

116 if attr == "seed":

117 msg = "Calling `.seed()` on instances is deprecated. " "Use the class method `Faker.seed()` instead."

--> 118 raise TypeError(msg)

119 else:

120 return super().\_\_getattribute\_\_(attr)

TypeError: Calling `.seed()` on instances is deprecated. Use the class method `Faker.seed()` instead.

ChatGPT: It seems there's an error with the faker library usage due to the deprecation of the .seed() method. Let's fix this by using the class method Faker.seed() instead. Here's the updated code:

pythonCopy code

import random

from faker import Faker

def generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=5, sentences\_per\_paragraph=5):

fake = Faker()

Faker.seed(42) # Setting seed for reproducibility

lorem\_ipsum\_text = ""

for \_ in range(num\_paragraphs):

paragraph = fake.paragraphs(nb=sentences\_per\_paragraph)

lorem\_ipsum\_text += "\n\n".join(paragraph) + "\n\n"

return lorem\_ipsum\_text

def save\_text\_to\_file(text, filename):

with open(filename, "w", encoding="utf-8") as file:

file.write(text)

def read\_text(filename):

with open(filename, 'r', encoding='utf-8') as file:

return file.read()

def tokenize\_text(text):

return text.split()

def create\_markov\_chain(tokens, chain\_length):

chain = {}

for i in range(len(tokens) - chain\_length):

prefix = tuple(tokens[i:i + chain\_length])

suffix = tokens[i + chain\_length]

if prefix in chain:

chain[prefix].append(suffix)

else:

chain[prefix] = [suffix]

return chain

def generate\_sentence(chain, start\_words, num\_generated):

sentence = list(start\_words)

current\_prefix = tuple(start\_words)

for \_ in range(num\_generated):

if current\_prefix not in chain:

break

next\_word = random.choice(chain[current\_prefix])

sentence.append(next\_word)

current\_prefix = tuple(sentence[-len(start\_words):])

return ' '.join(sentence)

def generate(filename, start\_words, chain\_length, num\_generated):

text = read\_text(filename)

tokens = tokenize\_text(text)

chain = create\_markov\_chain(tokens, chain\_length)

return generate\_sentence(chain, start\_words, num\_generated)

# Generate Lorem Ipsum text

lorem\_ipsum\_text = generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=10, sentences\_per\_paragraph=5)

# Save the text to a file

filename = "LoremIpsum.txt"

save\_text\_to\_file(lorem\_ipsum\_text, filename)

print(f"Lorem Ipsum text saved to '{filename}'")

# Example usage:

start\_words = ['Lorem', 'ipsum']

chain\_length = 2 # Adjust the chain length as needed

num\_generated = 50 # Adjust the number of words to generate as needed

generated\_sentence = generate(filename, start\_words, chain\_length, num\_generated)

print(generated\_sentence)

This should resolve the TypeError caused by the deprecated .seed() method usage.

—

Human: why is the output only Lorem Ipsum text saved to 'LoremIpsum.txt'

Lorem ipsum and not generating anything further

ChatGPT: The issue seems to arise from the way the Markov chain is generated and used. It appears that there's an error in how the Markov chain is constructed from the Lorem Ipsum text and then subsequently used for generating text.Let's make sure the Markov chain generation and text generation functions are working correctly. We'll print out some intermediate results for debugging purposes. Here's the updated code with additional print statements:

pythonCopy code

import random

from faker import Faker

def generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=5, sentences\_per\_paragraph=5):

fake = Faker()

Faker.seed(42) # Setting seed for reproducibility

lorem\_ipsum\_text = ""

for \_ in range(num\_paragraphs):

paragraph = fake.paragraphs(nb=sentences\_per\_paragraph)

lorem\_ipsum\_text += "\n\n".join(paragraph) + "\n\n"

return lorem\_ipsum\_text

def save\_text\_to\_file(text, filename):

with open(filename, "w", encoding="utf-8") as file:

file.write(text)

def read\_text(filename):

with open(filename, 'r', encoding='utf-8') as file:

return file.read()

def tokenize\_text(text):

return text.split()

def create\_markov\_chain(tokens, chain\_length):

chain = {}

for i in range(len(tokens) - chain\_length):

prefix = tuple(tokens[i:i + chain\_length])

suffix = tokens[i + chain\_length]

if prefix in chain:

chain[prefix].append(suffix)

else:

chain[prefix] = [suffix]

return chain

def generate\_sentence(chain, start\_words, num\_generated):

sentence = list(start\_words)

current\_prefix = tuple(start\_words)

for \_ in range(num\_generated):

if current\_prefix not in chain:

break

next\_word = random.choice(chain[current\_prefix])

sentence.append(next\_word)

current\_prefix = tuple(sentence[-len(start\_words):])

return ' '.join(sentence)

def generate(filename, start\_words, chain\_length, num\_generated):

text = read\_text(filename)

tokens = tokenize\_text(text)

chain = create\_markov\_chain(tokens, chain\_length)

# Debugging print statements

print("Tokens:", tokens)

print("Chain:", chain)

return generate\_sentence(chain, start\_words, num\_generated)

# Generate Lorem Ipsum text

lorem\_ipsum\_text = generate\_lorem\_ipsum\_paragraphs(num\_paragraphs=10, sentences\_per\_paragraph=5)

# Save the text to a file

filename = "LoremIpsum.txt"

save\_text\_to\_file(lorem\_ipsum\_text, filename)

print(f"Lorem Ipsum text saved to '{filename}'")

# Example usage:

start\_words = ['Lorem', 'ipsum']

chain\_length = 2 # Adjust the chain length as needed

num\_generated = 50 # Adjust the number of words to generate as needed

generated\_sentence = generate(filename, start\_words, chain\_length, num\_generated)

print("Generated sentence:", generated\_sentence)

By examining the output of the print statements, we can identify where the issue lies and adjust the code accordingly. Please run this code and provide the output for further diagnosis.

—

Human: It works fine but wouldn't it be better to generate random words rather than using for loops

ChatGPT: Generating random words directly instead of using a loop to select them from the Markov chain might introduce some challenges because Markov chains typically rely on the frequency of words or sequences of words in the input text. Simply choosing random words might not produce text that resembles the input text.However, if you want to generate random text without relying on the Markov chain, you can use alternative methods like sampling from a probability distribution of words or using a language model to generate text. Here's how you could modify the code to generate random words without relying on the Markov chain:

pythonCopy code

import random

import string

def generate\_random\_word():

# Generate a random word of random length between 1 and 10 characters

word\_length = random.randint(1, 10)

return ''.join(random.choice(string.ascii\_lowercase) for \_ in range(word\_length))

def generate\_random\_sentence(num\_words):

# Generate a sentence with the specified number of random words

return ' '.join(generate\_random\_word() for \_ in range(num\_words))

# Example usage:

num\_words = 50 # Adjust the number of words to generate as needed

random\_sentence = generate\_random\_sentence(num\_words)

print(random\_sentence)

This approach generates random words of random lengths and then combines them into a sentence. However, please note that this will not produce text similar to the input text, as it completely ignores the structure and semantics of the input. If you need text generation that resembles the input text, sticking with the Markov chain approach or using more advanced language models would be more appropriate.

—

Human: ohkay but if the n number of start words are given then generate sentence

ChatGPT: If you want to generate a sentence starting with a given set of start words while still using random word generation, you can modify the code to start the sentence with those words and then continue generating random words. Here's how you can do it:

pythonCopy code

import random

import string

def generate\_random\_word():

# Generate a random word of random length between 1 and 10 characters

word\_length = random.randint(1, 10)

return ''.join(random.choice(string.ascii\_lowercase) for \_ in range(word\_length))

def generate\_sentence\_with\_start\_words(start\_words, num\_words):

# Start the sentence with the given start words

sentence = start\_words[:]

# Generate additional random words to complete the sentence

remaining\_words = num\_words - len(start\_words)

if remaining\_words > 0:

sentence.extend(generate\_random\_word() for \_ in range(remaining\_words))

return ' '.join(sentence)

# Example usage:

start\_words = ['The', 'quick', 'brown']

num\_words = 10 # Adjust the total number of words in the sentence as needed

random\_sentence = generate\_sentence\_with\_start\_words(start\_words, num\_words)

print(random\_sentence)

In this code, the generate\_sentence\_with\_start\_words function takes a list of start words and a total number of words to generate. It starts the sentence with the given start words and then generates additional random words to complete the sentence if needed. Adjust the start\_words list and num\_words variable according to your requirements.

—

Human: Do the testing of this complete program with random function given some start words with a testing strategy of 15 cases

ChatGPT: Certainly! Here's how we can test the complete program with different sets of start words and various lengths of sentences:

pythonCopy code

def test\_random\_sentence\_generation():

test\_cases = [

(['The'], 5), # Start with one word

(['The', 'quick'], 10), # Start with two words

(['The', 'quick', 'brown'], 7), # Start with three words

(['Hello'], 3), # Start with one word

(['Hello', 'world'], 8), # Start with two words

(['This', 'is', 'a', 'test'], 6), # Start with four words

(['Lorem'], 4), # Start with one word

(['Lorem', 'ipsum'], 9), # Start with two words

(['Lorem', 'ipsum', 'dolor'], 12), # Start with three words

(['Python'], 5), # Start with one word

(['Python', 'programming'], 10), # Start with two words

(['Artificial', 'intelligence'], 7), # Start with two words

(['OpenAI'], 3), # Start with one word

(['Natural', 'Language', 'Processing'], 8), # Start with three words

(['Deep', 'Learning', 'models', 'are'], 6), # Start with four words

]

for i, (start\_words, num\_words) in enumerate(test\_cases, start=1):

print(f"Test Case {i}:")

print("Start Words:", start\_words)

print("Number of Words:", num\_words)

random\_sentence = generate\_sentence\_with\_start\_words(start\_words, num\_words)

print("Generated Sentence:", random\_sentence)

print()

# Perform testing

test\_random\_sentence\_generation()

This testing strategy includes 15 test cases with various combinations of start words and sentence lengths. It verifies that the function can correctly generate sentences with different starting conditions. You can expand this testing strategy further by adding more test cases if needed.

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_