Report, Graded Assignment 1 – DTE-2501

Task 1 – Interpretation

When I was told of the assignment earlier this autumn, I was told we were going to make a Small Language Model. Regarding this, I was expecting something more in the lines of what I have later learned is RNN or Transformer models, where you could query the model and get a response. What we are creating when using Markov Chain is basically a model that creates random sentences based on probability of a word following another, and a seed word to start the sentence with. I am writing this after creating the model, my plan was originally to make a CLI, but due to issues with the pynput-library, I ended up creating a GUI with ttkbootstrap instead.

Task 2 – Dataset

1. I have used Your Mom jokes as the only document in my corpus, which can be found here: https://bestlifeonline.com/yo-mama-jokes/
   This is an easy and simple way of getting text that made some sense. I initially tried using all three Lord of the Rings books as well as all three Fifty Shades books. This gave some unexpected results, as the sentence structure and word choice of each author varies wildly from each other. Also, there was too much data with Lord of the Rings totaling in at 980 pages in Norwegian, which I assume the English version also is close to. Due to this, and the obvious copyright issues I might face, I have chosen to not include these in my code.
2. My preprocessing has a fair bit of steps, some are obvious legacy from when I used LOTR and Fifty Shades, where I go through the abbreviations like Mr., Ms. and so forth:

```
abbreviations = ["Mr.", "Mrs.", "Miss.", "Dr.", "e.g.", "i.e.", "etc."]
for abbreviation in abbreviations:
                content = content.replace(abbreviation,
abbreviation.replace('.', '__PERIOD__'))
for abbreviation in abbreviations:
                self.sentences =
[sentence.replace(abbreviation.replace('.', '__PERIOD__'), abbreviation)
for sentence in self.sentences]
```

   I also remove all contractions like I'm, I've and similar:

```
content = replace_contractions(content)
```

   The following is the main preprocessing and sentence splitting functionality:

```
content = re.sub(r'[^\w\s\n—–]+´`', ' ', content)
content = re.sub(r'[\d,”“]', '', content)
self.sentences = re.split(r'[.!?]', content)
for sentence in self.sentences:
    self.sentence_objects.append(Sentence(sentence, source=self.name))
```

   As can be seen above, I am storing the sentences in a class called Sentence, which a Document object contains multiple of. These Document objects are then stored in a Corpus object, which contains zero to N documents.

Task 3 – The model

I am using a dictionary in my MarkovChain class which holds the Markov Chain, this consists of WordState objects, which in turn contains a dictionary of words following the root word and their occurrence rate. I am using a "#" as the start word, as in the example, so you can generate from "nothing", so to speak, but the default seed word is "your" since it is a joke generator after all. If you set it to blank it will default to "#" automatically, so it generates random start words as well.

To do this I iterate through every sentence in the corpus and create WordState objects from these words if they do not exist already.

```
for sentence in sentences:
       words = sentence.split()
       prev_word = "#"

       for word in words:
           if prev_word not in self.markov_chain:
               self.markov_chain[prev_word] = WordState()

           self.markov_chain[prev_word].add_next_word(word)
           prev_word = word
for current_word, word_state in self.markov_chain.items():
    total_transitions = sum(word_state._next_words.values())
    self.markov_chain[current_word]._next_words = {word: count / total_transitions
for word, count in word_state._next_words.items()}
```

Task 4 – Generate Text

To generate text I am using my MarkovChain class to generate based on seed word and sentence length, both of which can be changed in the settings menu.

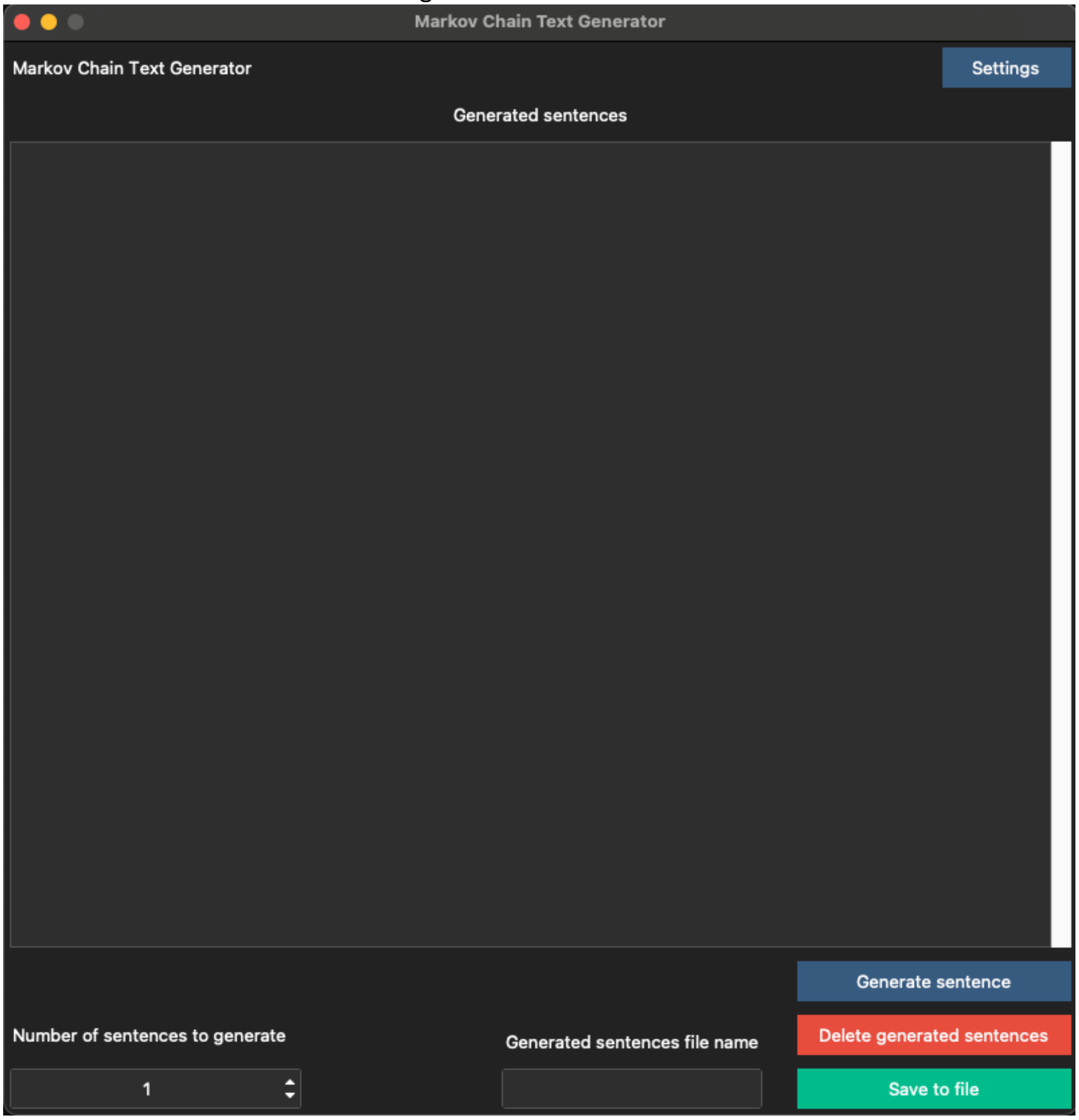
```
    def generate_text(self, seed=None, length=50):
        current_token = seed if seed is not None and seed != "#" else
random.choice([key for key in self.markov_chain.keys() if key != "#"])
        text = [current_token]

        for _ in range(length):
            try:
                next_word_state = self.markov_chain.get(current_token)
                if next_word_state is None:
                    break
                next_token = next_word_state.get_next()
                if next_token is None:
                    break
                text.append(next_token)
                current_token = next_token
            except KeyError:
                print(f"Key '{current_token}' not found in Markov chain. Stopping
text generation.")
                break
        self.generated_sentences.append(' '.join(text))
```

Task 5 – User Interface

For my interface, I had originally made a CLI using pynput to take in keypresses, but I ran into issues with the listener, if I wanted to add more documents to the corpus, I had to disable the listener, inside the onkey function that tracked which key was being pressed. The problem I ran into was that I had to have 3-4 nested if statements just to disable it and re-enable it where needed, as well as tracking all the keys in the appropriate places. If I did not do this, and a document name contained "q", it would just exit the program.

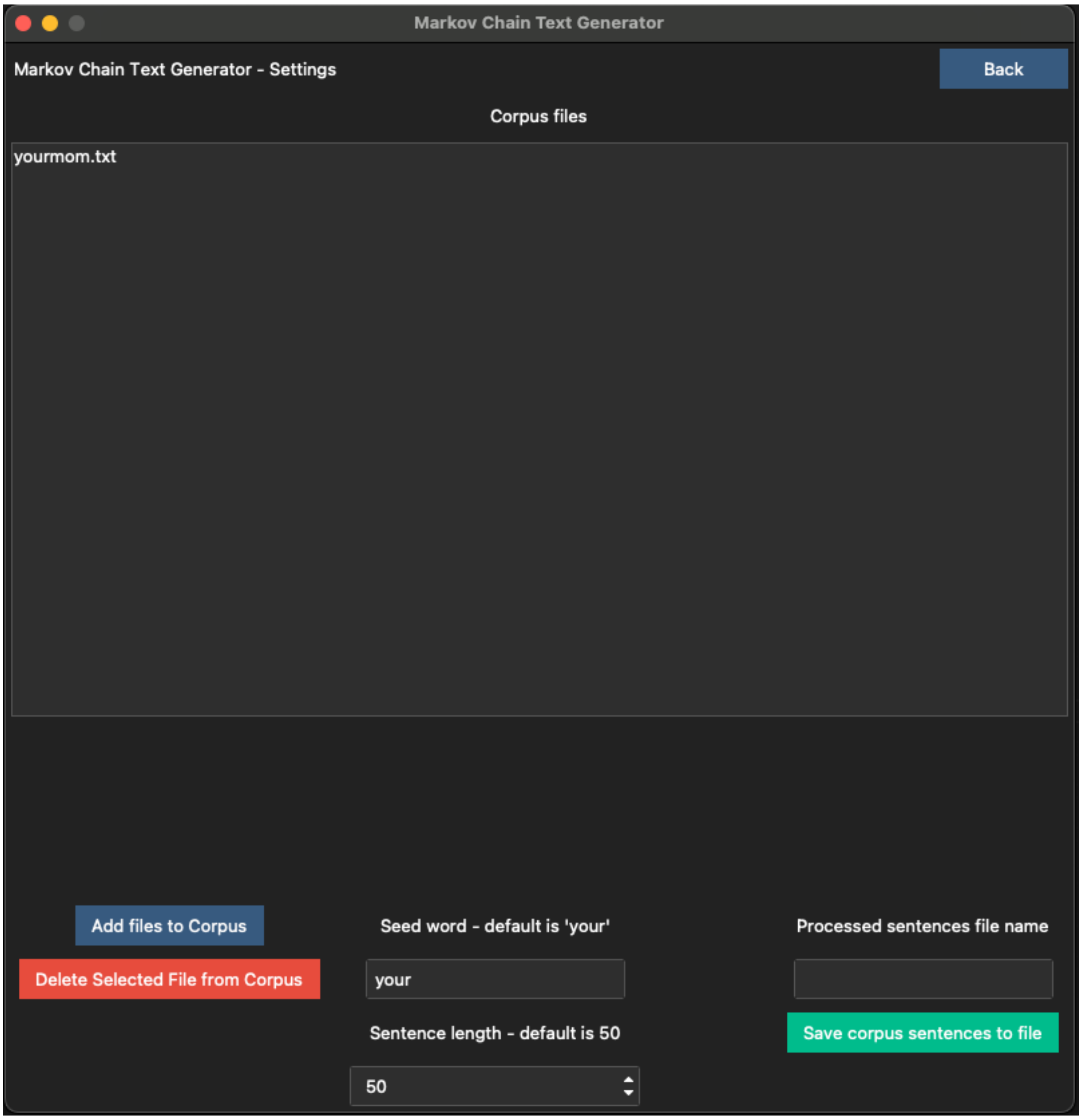
To combat this, I gave this idea up completely, and decided to join the rest of the world in 2023, and re-wrote everything to use Tkinter with ttkbootstrap, as Tkinter is more 2003 than 2023. Ttkbootstrap is a library that uses themed Tkinter (ttk), with Bootstrap features like colors, button styles and so forth.

The main interface looks as following:


Markov Chain Text Generator
Markov Chain Text Generator
Settings
Generated sentences
Generate sentence
Number of sentences to generate
1
Generated sentences file name
Delete generated sentences
Save to file

Here you can enter settings up top, change the number of sentences to generate, save the sentences to file, and delete all generated sentences (these are stored in a list so they are kept when going to settings and back). Here you can clearly see the Bootstrap at work, with the button colors being primary, success and danger.

It should be noted that there is a validation both for the number of sentences to generate, and the sentence length, neither can be negative, and will not throw errors if set to 0.


Markov Chain Text Generator
Markov Chain Text Generator - Settings
Back
Corpus files
Add files to Corpus
Delete Selected File from Corpus
Seed word - default is 'your'
Sentence length - default is 50
Processed sentences file name
Save corpus sentences to file

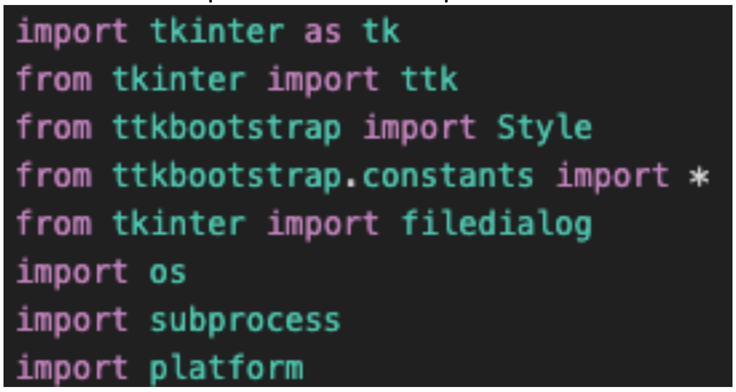
yourmom.txt

your

50

In the settings window, you can go back to the main window (which also saves any changes), add files to the corpus, save the corpus sentences to a file (it will append, not overwrite, if the file already exists!), delete a file from the corpus, you can also change the seed word and sentence length.

As for my imports, I have not made a requirements.txt, as it only needs to download Tkinter and ttkbootstrap. Here are the imports I have:



```python
import tkinter as tk
from tkinter import ttk
from ttkbootstrap import Style
from ttkbootstrap.constants import *
from tkinter import filedialog
import os
import subprocess
import platform
```