Project 4: Book Bot AI

CPSC 131 Fall 2019

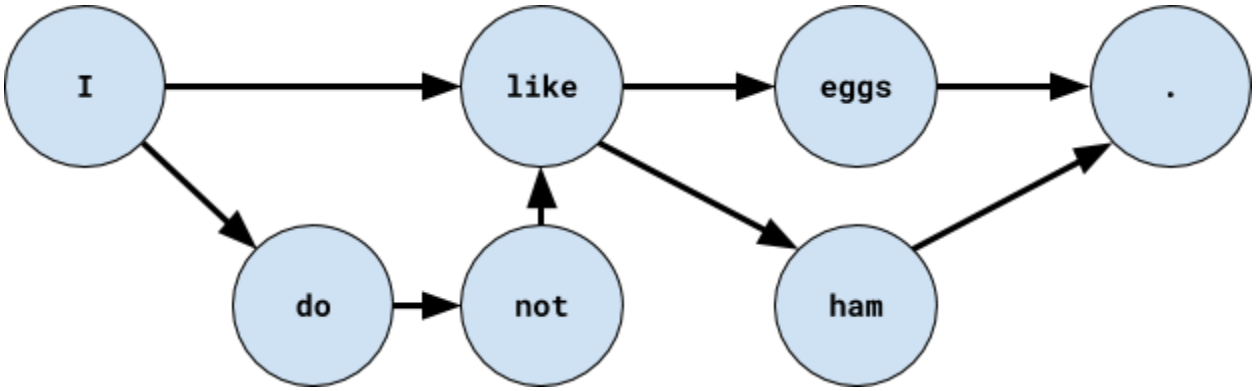
Introduction

You're tasked with working on an Artificial Intelligence that can generate convincing sentences. Your AI won't be too complicated; it will train itself on sentences from a given text file. It will form "Markov Chains" storing associations between two words. When generating a sentence it will move from one word to the next randomly.

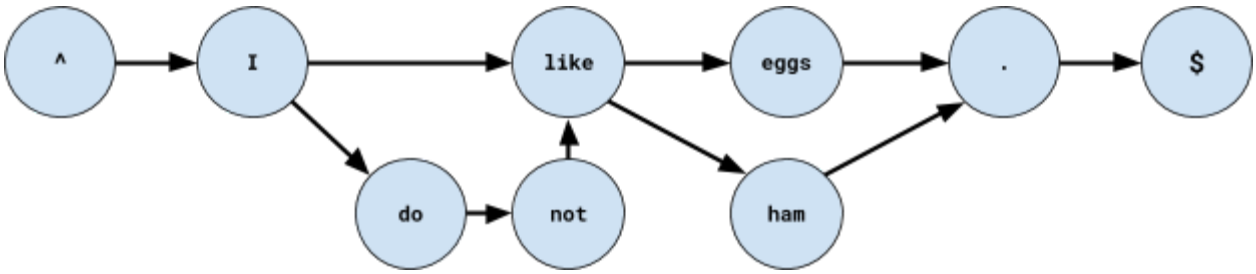
For example, given the following sentences:

```
I like eggs.
I do not like ham.
```

The Markov Chain generated could be expressed like so:



Expanded out with a starting and ending tokens:



With this structure, we can generate the following sentences, which never showed up originally:

```
I do not like eggs.
I like ham.
```

Objective

You are given a header file, BookBot.h, that defines what functions you need to implement. The main idea is to break sentences into words and store every word with its *following* word in a data structure. To generate a new sentence, you pick a word and then *randomly* pick one of its following word associations.

Complete the implementation of this class, adding public/private member variables and functions as needed. Your code is tested in the provided `main.cpp`.
You will need to implement the following functions:

- **BookBot(unsigned int seed)** - The constructor; given an unsigned int seed value, you are to seed the random number generator with the command `srand(seed)`. This ensures that the sentences you generate should match the ones we expect in the tests (as long as everything else works correctly).
- **`bool isEndPunctuation(char)`** - This should return true if the given char is one of the following: **`. ? !`**
- **`void read_in(const string &)`** - See hints.
- **`vector<string> getValues(const string &)`** - See hints.
- **`generateSentence()`** - See hints

Source Code Files

You are given "skeleton" code files with declarations that may be incomplete and without any implementation. Implement the code and ensure that all the tests in `main.cpp` pass successfully.

- `BookBot.h`: This is to be completed
- `BookBot.cpp`: This is to be completed
- `sanitize.h`: This contains helper functions you may wish to use.
- `main.cpp`: The main function tests the output of your functions. You may wish to add additional tests. During grading your main.cpp file will be replaced with the one you were provided with.
- `README.md`: You must edit this file to include your name and CSUF email. This information will be used so that we can enter your grades into Titanium.

Hints

From the earlier example of our Markov Chain, when listed out:

```
Key     Values
I     : { "like", "do" }
do    : { "not" }
not   : { "like" }
```

```
like  : { “eggs”, “ham” }
eggs  : { “.” }
ham   : { “.” }
```

We will store this in a **std::map**. Our keys will be the first word, the values will be a vector of words that follow.

We’ll expand this a bit further and have starting tokens at the front and end of each sentence. The beginning of a sentence will be denoted with **^** and the end of a sentence will be denoted with **$**. This way we have a consistent begin and end that is always the same. Our listed associations become:

```
Key     Values
^     : { “I” }
I     : { “like”, “do” }
do    : { “not” }
not   : { “like” }
like  : { “eggs”, “ham” }
eggs  : { “.” }
ham   : { “.” }
.     : { “$” }
```

You do not need to implement sanitize(). You are provided a helper header file: sanitize.h, which contains some useful functions including sanitize.h. It also has a **debug()** function to print out your **map<string, vector<string>>**.

Make sure you keep track of the difference between a single char denoted with single quotes vs. a string stored with double quotes. You can index into a string using array notation, which will return a single char. For example:

```
string word = “Hello?”;
char punc = word[word.size() - 1];    // Stores ‘?’
```

You can separate out punctuation from a word by calling a string constructor and substr() function.

```
string punctuation(1, word[word.size() - 1]);
word = word.substr(0, word.size() - 1);
```

The first line is constructing a string, one character in length, with the last character of the word. The second line is creating a substring starting from the very first character (index 0) and with a length of the word size - 1. This will chop off the last letter.

- **`void read_in(const string &)`** - This should open up the given file and read in a single word at a time, “sanitize” it (remove non-alphabetic characters like dashes, slashes, etc.), and check if it ends in punctuation.
    - If it ends with punctuation:
        i. Separate the end punctuation from the new word you just read in.
        ii. Store the old word as a key, and add the new word you just read in without the end punctuation.
        iii. You will need to store the new word you read in without the end punctuation as a key, and add the punctuation as a value.
    - If it doesn’t end with punctuation and it’s not an empty string, store the old word with the new word you just read in.
    - Finally make sure that any end punctuation as a key has an associated value of **$**
- **`vector<string> getValues(const string &)`** - This should return the vector of words associated with a given key.
- **`string generateSentence()`** - This should return a randomly generated sentence. Start your sentence generation with the **^** key. From the associated vector, randomly pick one token word, and add it to your sentence result (separated by a space “ “). Call the C++ function `rand()` to get a random number. Use your token word to grab the next list of candidates and repeat until the token word you get is **$**

Obtaining and submitting code

We will be using GitHub Classroom to distribute the skeleton code and collect your submissions. Click the assignment link to fork your own copy of the skeleton code to your PC.

https://classroom.github.com/a/Ynnle3Hc

Testing

Unless otherwise directed, use the following command to compile your program:
`clang++ -g -std=c++14 main.cpp BookBot.cpp -o test`

To attempt to run the compiled test program, use the following command:
`./test`

Grading rubric

Your grade will be comprised of two parts, *Form* and *Function*.
Function refers to whether your code works properly as tested by the main function (80%).
Form refers to the design, organization, and presentation of your code. An instructor will read your code and evaluate these aspects of your submission (20%).

Deadline

The project deadline is **December 1, 2019** 11:55pm.

You will be graded based on what you have pushed to the main branch of your GitHub repository as of the deadline. Commits made after the deadline will not be considered. Late submissions will not be accepted.

Your code must compile/build for it to be tested and graded. If you only complete part of the project, make sure that it compiles before submitting.

Blurb for your resume

Use your GitHub account as a ready-to-show portfolio of your programming projects to potential employers. You can also add this blurb to your resume after successfully completing this project:

> *Created a C++ program that generates random sentences using Markov chains after being trained on classic books.*

Credits

This project includes two text files from Project Gutenberg.