CSC108: Lazy Literature

Daniel Zingaro
(based on an assignment by Danny Heap)
University of Toronto
daniel.zingaro@utoronto.ca

October 2014

1 Introduction

In the next couple lectures, we're going to write a larger program that uses dictionaries. The program will work in two phases: in the first phase, it will consume a text (like Alice in Wonderland) and remember information about that text by storing information in a dictionary. In the second phase, the program will try to generate text in the style of the text that it consumed, by using the information that it earlier stored in the dictionary. The goal is therefore to generate new text that sounds like it was written by the author of the original text. The program is called Lazy Literature!

2 Phase 1: Learn

In order to mimic text, we first want to learn something about that text. Assume that in the entire text, the word `Dan` is always followed by one of three words: `played`, `walked`, or `said`. (For example, one of the sentences in the book could be "Dan walked slowly, contemplating the music to play in tomorrow's lecture.") What we want to store is this information: given the word `Dan`, what are the possible words that could follow `Dan`? That is, what are the words that the author used immediately after `Dan`? We could store this information in several ways, such as:

```
['Dan', ['played', 'walked', 'said']]
```

The reason we want to store this is so that in phase 2, we know our options for what can follow `Dan` and can choose one of them to continue the text.

However, we don't just want to store the words that follow `Dan`. We want to store the words that follow every possible word in the book, and want to be able to look up these words to find their associated values since the values give us our options for continuing the text. This type of lookup is much more easily done using a dictionary than a nested list as above. We will therefore use a dictionary to store this information, where each key is a word in the book, and each value is the list of words that follows the key word. For the

above example, we'd have:

```
{'Dan': ['played', 'walked', 'said']}
```

Let's now consider a text (in this case just one sentence) where we end up with some values whose lists contain more than one element. Assume that the text of our book is the following:

```
Did you get the sword from the old man on top of the waterfall?
```

Here is the dictionary that would result from processing this text.

```
{'': ['Did'], 'Did': ['you'], 'you': ['get'],
 'get': ['the'], 'the': ['sword', 'old', 'waterfall?'],
 'sword': ['from'], 'from': ['the'], 'old': ['man'],
 'man': ['on'], 'on': ['top'], 'top': ['of'],
 'of': ['the']}
```

The first word in this text is `Did`, and this is indicated in the dictionary by the empty string key associated with `Did`. Note also in the dictionary that all values are lists, and that the lists can have multiple elements (see `the`, which has three words in its associated list).

3 Phase 2: Generate

Once the dictionary is built, we can use it to generate text that mimics the style of the original. The way to do this is to start with the empty string, look up the empty string in the dictionary to find the first word, look up that word in the dictionary to find the next word, and so on. Whenever the dictionary contains multiple follow-up words (like for `the` above), one should be chosen randomly.

Assume we want to generate texts of five words. Here are the texts that can be generated from the above example:

```
Did you get the old
Did you get the waterfall?
Did you get the sword
```

There are only three possibilities here, but in huge texts (that we will use) there will be many. Also note that we chose five words here for a reason. Had we chosen six words, there is a possibility of getting an error if we are not careful. The problem is that `waterfall?` has no follow-up word (`waterfall?` is not a word in the dictionary).

4 Other Issues

- What if the word `dog` were followed in the original text by `barked` ten times and `food` twice? We'd like to prefer to choose `barked` most of the time, since that's what was done by the original author. How can we do this?

- If we start with `''` every time, our "story" will always start with the first word of the story on which ours was based. How can we start at a random word?
- We'll have to fix that `waterfall?` problem above. What should we do when we hit the last word of the text (which has no follow-up words)? It's unlikely to happen to us in large texts, but it is still a bug and must be fixed!
- Above, we used only one word (the current word) to determine the word that should follow. But what if we used more context to decide what word to choose next? If we took more text into account, our follow-up words might be more reasonable. For example, we could use the most recent three words to decide the next word.

5 Development

We will design one major function for each of the two phases. First, we'll write a function `make_dictionary` that takes a text file and produces the dictionary that maps words to lists of follow-up words. Then, we'll write a corresponding function `mimic_text` that takes such a dictionary and uses it to produce a new text that mimics the style of the original.