Lab 10: Markov Chains

CSE/IT 107

NMT Computer Science

"Holy shit, you geeks are badass."

— Pam (*Archer*)

"Simplicity is prerequisite for reliability."

— Edsger W. Dijkstra

"Simplicity is the final achievement. After one has played a vast quantity of notes and more notes, it is simplicity that emerges as the crowning reward of art."

— Frèdèric Chopin

"The truth is a trap: you can not get it without it getting you; you cannot get the truth by capturing it, only by its capturing you."

— Søren Kierkegaard

---

1 Introduction

Today's lab will be just another small project. We will not be covering any new topics: this is primarily an application of what you already know. You will be writing a program that generates Markov chains from an input file. Markov chains are the simplest way for you to generate sentences that almost make sense, but really don't. They are based on figuring out the likelihood of a word following another word by looking at existing bodies of text (for example, Wikipedia). Then, to generate sentences you choose a starting word and based on a random variable as well as the probabilities that you've found by looking at existing text, you choose a word following that starting word and repeat.

This will generate sentences like this when used on conversations between Russell and Chris about Python lab (the starting word was "CS"):

> "CS be hard!"
>
> — arcbot

Garkov: A Garfield comic generated using Markov chains.

2 Markov Chains

A Markov chain is a method of randomly generating a sequence based on a set of input data. In this lab, we will be using Markov chains to generate sentences based on an input text file. In order to do this, we must understand how Markov chains work.

The basic steps of creating Markov chains are:

1. Select a random starting word to start our new sentence.
2. From all the words that ever follow that word in the input sequence, choose one. Add that word to the end of our new sentence.
3. Continue selecting randomly from the words that can possibly follow the current last word of our sentence until either there are no possible choices or we have made a sentence as long as desired.

For a simple example, let's generate Markov phrases using inputs of "Hello, how are you?" and "Where are my keys?". If we convert these sentences into a graph showing the possible results, we would get Figure 1.

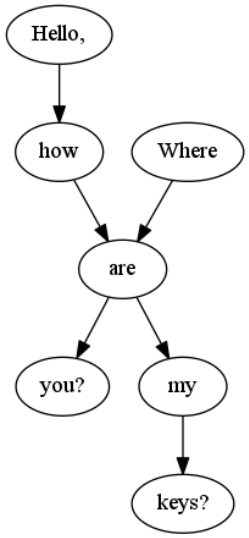

Figure 1: A graphical representation of the Markov possibilities for "Hello, how are you?" and "Where are my keys?"

In this graph, each arrow represents a choice we can take based on the last word we added to our sentence, continuing until there are no valid paths to take. Looking at the graph, it's pretty easy to see there are four possible outputs if we start our chain with either "Hello," or "Where":

- Hello, how are you?
- Hello, how are my keys?
- Where are you?
- Where are my keys?

For a more complex example, let's use the input phrase "There is a fifth dimension, beyond that which is known to man. It is a dimension as vast as space and as timeless as infinity.". If we were to convert this sentence into a graph representing the possible choices to make at each step, it would look something like Figure 2.

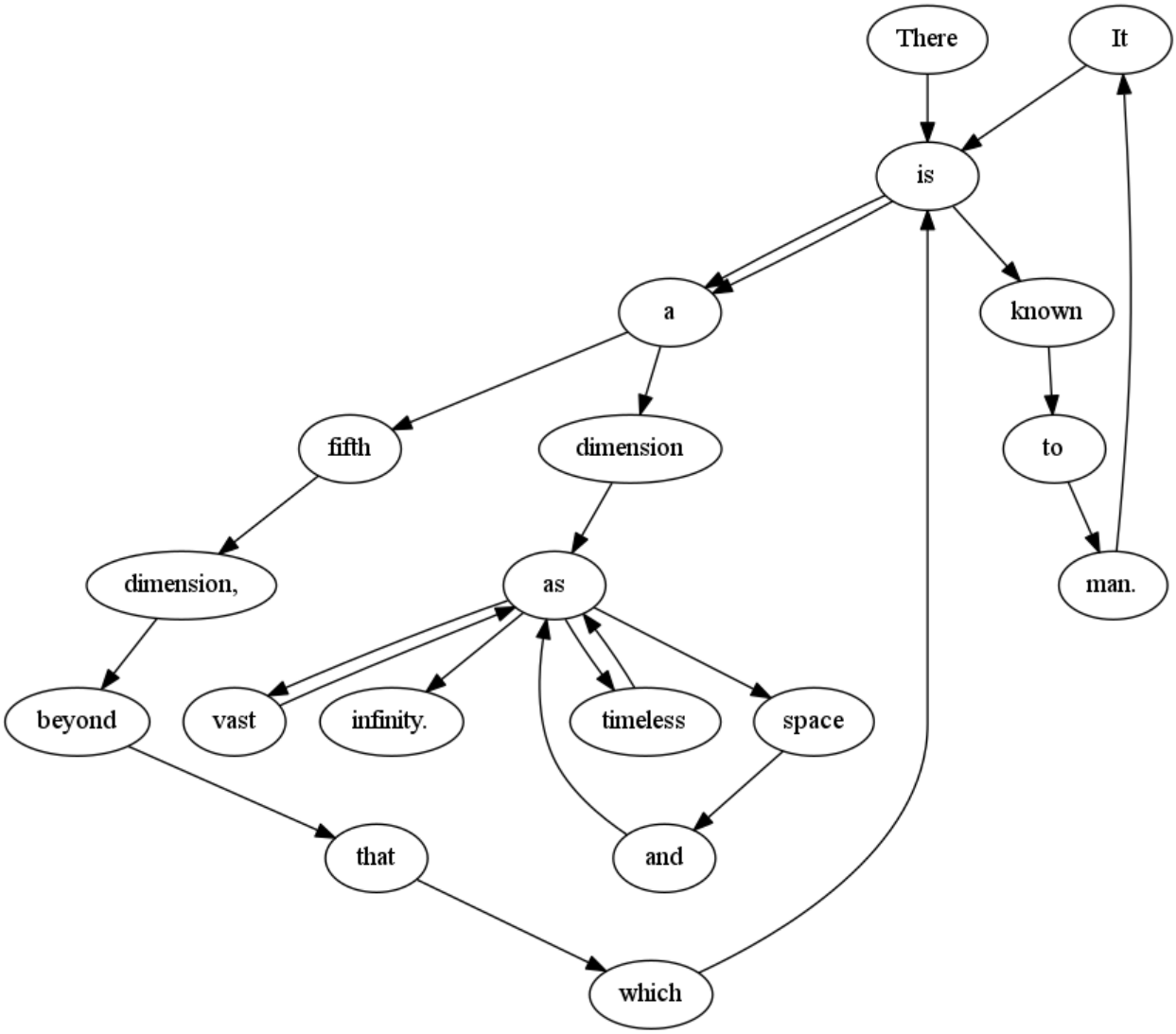

Figure 2: A graphical representation of the Markov possibilities for *The Twilight Zone*'s introduction.

In this graph, if we start with "There", our only option for a next word is "is". "is" is followed by "a" twice and "known" once, so it has two arrows to "a" and one to "known". This means that, when we randomly choose a next step, we should have a $\frac{2}{3}$ chance to choose "a" and a $\frac{1}{3}$ chance to choose "known". If we (randomly) choose to continue to "a", we have the choice of either "fifth" or "dimension" to continue our sentence with. A few possible new sentences we could generate from this input are:

- There is a dimension as infinity.
- There is a fifth dimension, beyond that which is a fifth dimension, beyond that which is a dimension as vast as space and as infinity.

- There is known to man. It is known to man. It is a dimension as space and as infinity.

As you can see, Markov chains have a tendency to make sentences which almost make sense. This is because every individual pairing of two words will make sense, but the combinations of the pairings might not. For example, "as vast" and "vast as" can both make sense given the right context, but "vast as vast as vast as vast" is nonsense. We can help alleviate this problem by taking into account the last 2 (or 3, or 4...) words when choosing the next word instead of just the last one, but this requires a far larger input or it will result in the output being the same as the input.

2.1 Random Numbers

You may find the module `random` to be especially useful when creating Markov chains. Below is a brief example of some common functions inside `random`.

```
>>> import random
>>> print(random.random())
0.7682548548225483
>>> print(random.uniform(1, 100))
36.30623079969581
>>> print(random.choice(['joe', 'moe', 'larry', 'shemp', 'curly']))
larry
>>> print(random.randint(1, 100))
79
```

Take a look at the documentation for `random` if you would like a random number not based on a uniform distribution:

`https://docs.python.org/3.4/library/random.html`

Function	Arguments	Purpose
`random.random()`		Returns a random number between 0.0 and 1.0, including 0.0 but not 1.0.
`random.uniform()`	`a, b`	Returns a random number between `a` and `b` inclusive. Each real number between `a` and `b` has an equal probability of occurring.
`random.randint()`	`a, b`	Returns a random integer between `a` and `b` inclusive. Each integer between `a` and `b` has an equal probability of occuring.
`random.choice()`	`list`	Returns a random value from `list`.

Table 1: Summary of `random` functions.

3 Amnesty

In honor of getting through 10 labs, we are allowing you to resubmit one previous lab for a new grade. If you have a previous lab that you would like to improve your grade on, you may rewrite that lab and upload a new version to be graded. This new grade, if better than your original (hopefully we can assume that will be the case) will replace your original grade.

Upload your redone assignment to the "Amnesty" assignment on Canvas. Make sure it is clear from your filename which lab you are redoing.

4 Exercises

Boilerplate

Remember that this lab *must* use the boilerplate syntax introduced in Lab 5.

markov.py Write a program that takes in a filename, reads each line of the file, converts the lines into a format convenient for making Markov chains, and then prints out a new sentence randomly generated from the data, based on the Markov algorithm. When loading the file, treat the lines as separate statements. That is, if "Hello, how are you?" and "Where are my keys?" are lines in a file, then "you?" should not be followed by "Where" when generating a chain. However, "are" should be allowed to be followed by either "you?" or "my", as seen in Figure 1.

When creating a new chain, the first element should always be a randomly selected first word of a line in the file. The chain should end when either:

- There are no valid choices to continue the sentence with.
- The sentence has reached a length of 100 words.

Remember that you need to account for different frequencies of possible follow words. That is, a situation like in Figure 2 where there is a higher chance for "a" to follow "is" than for "known" to. You need to account for whatever probabilities might come up within your input file.

You may use any file you wish for test input, though `reviews.txt` is provided for you. This is a collection of 12500 reviews from IMDB and is a subset of the data provided at `http://ai.stanford.edu/~amaas/data/sentiment/index.html`. When parsing the input file, you should skip any blank lines.

Hint: You may find dictionaries to be useful in implementing Markov chains.

5 Submitting

Files to submit:

- markov.py (Section 4)
- Amnesty Lab (Section 3)

You may submit your code as either a tarball (instructions below) or as a .zip file. Either one should contain all files used in the exercises for this lab. The submitted file should be named either `cse107_firstname_lastname_lab10.zip` or `cse107_firstname_lastname_lab10.tar.gz` depending on which method you used.

For Windows, use a tool you like to create a `.zip` file. The TCC computers should have `7z` installed. For Linux, look at lab 1 for instructions on how to create a tarball or use the "Archive Manager" graphical tool.

Upload your tarball or .zip file to Canvas.