Markov Text Generation

COMP 2210 Assignment

Problem Description

The Infinite Monkey Theorem[1] (IFT) says that if a monkey hits keys at random on a typewriter it will almost surely, given an infinite amount of time, produce a chosen text (like the Declaration of Independence, *Hamlet*, or a script for ... *Planet of the Apes*). The probability of this actually happening is, of course, very small but the IFT claims that it is still possible. Some people have tested this hypotheis in software and, after billions and billions of simulated years, one virtual monkey was able to type out a sequence of 19 letters that can be found in Shakespeare's *The Two Gentlemen of Verona.* (See the April 9, 2007 edition of *The New Yorker* if you're interested; but, hypothesis testing with real monkeys[2] is far more entertaining.)

The IFT might lead to some interesting conversations with Rust Cohle, but the practical applications are few. It does, however, bring up the idea of *automated text generation*, and there the ideas and applications are not only interesting but also important. Claude Shannon essentially founded the field of information theory with the publication of his landmark paper *A Mathematical Theory of Computation*[3] in 1948. Shannon described a method for using Markov chains to produce a reasonable imitation of a known text with sometimes startling results. For example, here is a sample of text generated from a Markov model of the script for the 1967 movie *Planet of the Apes.*

```
"PLANET OF THE APES"
Screenplay by Michael Wilson
Based on Novel By Pierre Boulle

DISSOLVE TO: 138 EXT. GROVE OF FRUIT TREES - ESTABLISHING SHOT - DAY
Zira run back to the front of Taylor.
The President, I believe the prosecutor's charge of this man.
ZIRA  Well, whoever owned them was in pretty bad shape.
He picks up two of the strain.
You got what you wanted, kid. How does it taste?   Silence.
Taylor and cuffs him.
Over this we HEAR from a distance is a crude horse-drawn wagon
is silhouetted-against the trunks and branches of great trees and
bushes on the horse's rump. Taylor lifts his right arm to
ward off the blow, and the room and lands at the feet of Cornelius
and Lucius are sorting out equipment falls to his knees, buries
his head silently at the Ranch).
DISSOLVE TO: 197 INT. CAGES - CLOSE SHOT -
FEATURING LANDON - FROM TAYLOR'S VOICE (o.s.)
I've got a fine veternary surgeons under my direction?
ZIRA  Taylor!
ZIRA  There is a   small lake, looking like a politician.
TAYLOR  Dodge takes a pen and notebook from the half-open door
of a guard room. Taylor bursts suddenly confronted by his
```

[1] https://en.wikipedia.org/wiki/Infinite_monkey_theorem

[2] https://web.archive.org/web/20130120215600/http://www.vivaria.net/experiments/notes/publication/NOTES_EN.pdf

[3] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=6773024

```
original pursuer (the dismounted cop coming up with a cigar butt and
places it in the drawer beside them.
TAYLOR  What’s the best there is a. loud RAP at the doll was found
beside the building. Zira waits at the third table.
TAYLOR     Good question. Is he a man?
CORNELIUS     (impatiently.
DODGE    Blessed are the vegetation.
These SHOTS are   INTERCUT with:  94 WHAT THE ASTRONAUTS
They examine the remnants of the cage.
ZIRA (plunging on)   Their speech organs are adequate.
The flaw lies not in anatomy but in the back of his left sleeve.
TAYLOR (taking off his shirt.  80 DODGE AND LANDON
You don’t sound happy in your work.
GALEN  (defensively)  Gorilla hunter stands over a dead man, one fo
```

Besides a few spelling errors and some rather odd things that make you wonder about the author, this passage is surprisingly human-like. This is a simple example of *natural language generation*, a sub-area of *natural language processing*—a very active area of research in computer science. The particular approach we’re using in this assignment was famously implemented as the fictitious Mark V. Shaney[4] and the Emacs command Disassociated Press[5].

Approach

So, here’s the basic idea: Imagine taking a book (say, *Tom Sawyer*) and determining the probability with which each character occurs. You would probably find that spaces are the most common, that the character ‘e’ is fairly common, and that the character ‘q’ is rather uncommon. After completing this “level 0” analysis, you would be able to produce random Tom Sawyer text based on character probabilities. It wouldn’t have much in common with the real thing, but at least the characters would tend to occur in the proper proportion. In fact, here’s an example of what you might produce:

Level 0

```
rla bsht eS ststofo hhfosdsdewno oe wee h .mr ae irii ela iad o r te u t mnyto onmalysnce,
ifu en c fDwn oee iteo
```

Now imagine doing a slightly more sophisticated *level 1* analysis by determining the probability with which each character follows every other character. You would probably discover that ‘h’ follows ‘t’ more frequently than ‘x’ does, and you would probably discover that a space follows ‘.’ more frequently than ‘,’ does. You could now produce some randomly generated *Tom Sawyer* text by picking a character to begin with and then always choosing the next character based on the previous one and the probabilities revealed by the analysis. Here’s an example:

Level 1

```
"Shand tucthiney m?" le ollds mind Theybooure He, he s whit Pereg lenigabo Jodind alllld
ashanthe ainofevids tre lin-p asto oun theanthadomoere
```

Now imagine doing a *level k* analysis by determining the probability with which each character follows every possible sequence of characters of length k (*kgrams*). A level 5 analysis of *Tom Sawyer* for example, would reveal that ‘r’ follows “Sawye” more frequently than any other character. After a level k analysis, you would be able to produce random *Tom Sawyer* by always choosing the next character based on the previous k characters (a kgram) and the probabilities revealed by the analysis.

[4] https://en.wikipedia.org/wiki/Mark_V._Shaney

[5] https://en.wikipedia.org/wiki/Dissociated_press

At only a moderate level of analysis (say, levels 5-7), the randomly generated text begins to take on many of the characteristics of the source text. It probably won't make complete sense, but you'll be able to tell that it was derived from *Tom Sawyer* as opposed to, say, *The Sound and the Fury.*

Here are some more examples of text that is generated from increasing levels of analysis of *Tom Sawyer.* (These "levels of analysis" are called *order K Markov models.*)

K = 2

```
"Yess been." for gothin, Tome oso; ing, in to weliss of an'te cle - armit.  Papper a comeasione,
and smomenty, fropeck hinticer, sid, a was Tom, be suck tied.  He sis tred a youck to themen
```

K = 4

```
en themself, Mr.  Welshman, but him awoke, the balmy shore.  I'll give him that he couple
overy because in the slated snufflindeed structure's kind was rath.  She said that the wound
the door a fever eyes that WITH him.
```

K = 6

```
people had eaten, leaving.  Come - didn't stand it better judgment; His hands and bury it
again, tramped herself!  She'd never would be.  He found her spite of anything the one was
a prime feature sunset, and hit upon that of the forever.
```

K = 8

```
look-a-here - I told you before, Joe.  I've heard a pin drop.  The stillness was complete,
how- ever, this is awful crime, beyond the village was sufficient.  He would be a good enough
to get that night, Tom and Becky.
```

K = 10

```
you understanding that they don't come around in the cave should get the word "beauteous"
was over-fondled, and that together" and decided that he might as we used to do - it's nobby
fun.  I'll learn you."
```

To create an order K Markov model of a given source text, you would need to identify all kgrams in the source text and associate with each kgram all the individual characters that follow it. This association or mapping must also capture the frequency with which a given character follows a given kgram. For example, suppose that $k = 2$ and the sample text is:

```
agggcagcgggcg
```

The Markov model would have to represent all the character strings of length two (2-grams) in the source text, and associate with them the characters that follow them, and in the correct proportion. The following table shows one way of representing this information.

kgram	Characters that follow
ag	gc
gg	gcgc
gc	agg
ca	g
cg	g

Once you have created an order K Markov model of a given source text, you can generate new text based on this model as follows.

1. Randomly pick k consecutive characters that appear in the sample text and use them as the initial kgram.

2. Append the kgram to the output text being generated.

3. Repeat the following steps until the output text is sufficiently long.

   (a) Select a character c that appears in the sample text based on the probability of that character following the current kgram.

   (b) Append this character to the output text.

   (c) Update the kgram by removing its first character and adding the character just chosen (c) as its last character.

If this process encounters a situation in which there are no characters to choose from (which can happen if the only occurrence of the current kgram is at the exact end of the source), simply pick a new kgram at random and continue.

As an example, suppose that $k = 2$ and the sample text is that from above:

```
agggcagcgggcg
```

Here are four different output text strings of length 10 that could have been the result of the process described above, using the first two characters ('ag') as the initial kgram.

```
agcggcagcg
aggcaggcgg
agggcaggcg
agcggcggca
```

For another example, suppose that $k = 2$ and the sample text is:

```
the three pirates charted that course the other day
```

Here is how the first three characters of new text might be generated:

- A two-character sequence is chosen at random to become the initial kgram. Let's suppose that "th" is chosen. So, kgram = `th` and output = `th`.

- The first character must be chosen based on the probability that it follows the kgram (currently "th") in the source. The source contains five occurrences of "th". Three times it is followed by 'e', once it is followed by 'r', and once it is followed by 'a'. Thus, the next character must be chosen so that there is a 3/5 chance that an 'e' will be chosen, a 1/5 chance that an 'r' will be chosen, and a 1/5 chance that an 'a' will be chosen. Let's suppose that we choose an 'e' this time. So, kgram = `he` and output = `the`.

- The next character must be chosen based on the probability that it follows the kgram (currently "he") in the source. The source contains three occurrences of "he". Twice it is followed by a space and once it is followed by 'r'. Thus, the next character must be chosen so that there is a 2/3 chance that a space will be chosen and a 1/3 chance that an 'r' will be chosen. Let's suppose that we choose an 'r' this time. So, kgram = `er` and output = `ther`.

- The next character must be chosen based on the probability that it follows the kgram (currently "er") in the source. The source contains only one occurrence of "er", and it is followed by a space. Thus, the next character must be a space. So, kgram = `r_` and output = `ther_`, where '_' represents a blank space.

Implementation Details

You are provided with two Java files that you must use to develop your solution: `MarkovModel.java` and `TextGenerator.java`.

The constructors of `MarkovModel` build the order-k model of the source text. You are required to represent the model with the provided HashMap field.

The main method of `TextGenerator` must process the following three command line arguments (in the `args` array):

- A non-negative integer *k*
- A non-negative integer *length*.
- The name of an input file *source* that contains more than *k* characters.

Your program must validate the command line arguments by making sure that *k* and *length* are non-negative and that *source* contains at least *k* characters and can be opened for reading. If any of the command line arguments are invalid, your program must write an informative error message to `System.out` and terminate. If there are not enough command line arguments, your program must write an informative error message to `System.out` and terminate.

With valid command line arguments, your program must use the methods of the `MarkovModel` class to create an order *k* Markov model of the sample text, select the initial kgram, and make each character selection. You must implement the `MarkovModel` methods according to description of the Markov modeling process in the section above.

A few sample texts have been provided, but Project Gutenberg (`http://www.gutenberg.org`) maintains a large collection of public domain literary works that you can use as source texts for fun and practice.

Acknowledgments

This assignment is based on the ideas of many people, Jon Bentley and Owen Astrachan in particular.