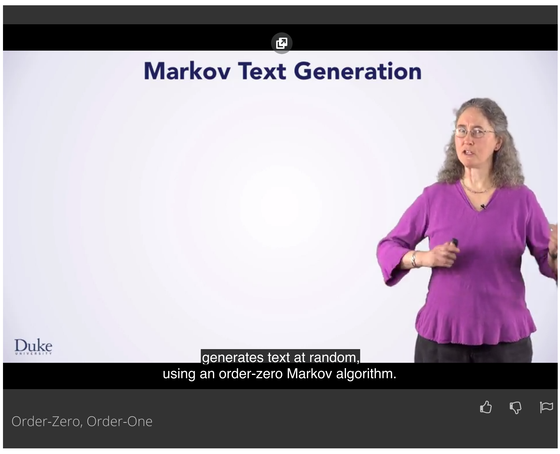
Generating Random Text



Have a question? Discuss this lecture in the week forums.

Interactive Transcript

Search Transcript English

0:03
Hi, in this lesson, we're going to walk through creating the class that generates text at random, using an order-zero Markov algorithm. We'll also describe what changes you'll need to make to create an order-one or more Markov class. Order-zero is straightforward to program, and will be a model for other Markov classes. All Markov models will use a training text as the basis for generating text at random. In an order-zero model, we don't use any characters to predict the next character. We choose every character at random from the entire training text. You can see an order-zero text here. Words are long, and letter combinations don't always make sense. It's hard to pronounce words like tmba, for example, or dnkmo. In an order-one Markov model, we choose a character based on one previous character. This makes letter combinations more common than an order-zero model as you may see here, where words like bests are created randomly, and words are easier to pronounce. Even where they are words like stesurd and tico. We'll provide a quick overview of developing the MarkovZero class. And you'll be able to use it to generate text randomly.

1:24
We'll think about methods first. Sometimes this is called the class' behavior. Thinking about methods will help as we think about what state or instance variables are needed.

1:37
We'll need to be able to set the training text for the MarkovZero class, and we'll need to be able to generate text randomly. That's two different methods.

1:47
We could combine these into one method, but in general, keeping methods single purpose is a good idea. In this case we might want to create several texts at random from this same training text. So keeping the methods separate makes lots of sense.

2:03
First we'll look at setting the training text in MarkovZero.

2:07
The training text is used when generating random text. As I mentioned earlier, we might want to create several texts at random from the same training text. This means we'll need to store the training text in an instance variable. The instance variable is assigned a value when the set training method is called, and then accessed when text is generated at random.

2:32
The other method we'll design and implement generates text at random. The getRandomText method will choose a character at random from the training text.

2:43
We'll use the nextInt method from the class java.util.random to create a random index, and we'll use this index to access the training text for a random character.

2:57
We'll create a StringBuilder object to store the random text, since adding or concatenating to a StringBuilder is efficient.

3:05
We'll append to the StringBuilder object, and we'll use the toString method to return a string when we're done.

3:14
We'll also need a constructor and perhaps other methods in the MarkovZero class.

3:21
Constructors typically initialize fields or instance variables.

3:26
In the MarkovZero class, we have an instance field that stores a random object from the java.util package.

3:34
We create a new random object in the constructor.

3:37
It's often useful to generate a reproducible sequence of random numbers to help with debugging. We can do that by setting what's called the seed of the random number generator, which we do here in creating a new random object. This may help as you debug the Markov classes.

3:57
There's also an instance field myText, that's not initialized in the constructor, but is by the setTrainingText method we discussed earlier.

4:07
This is enough background to write and test the MarkovZero class. But we'll provide some guidance for MarkovOne as well.

4:16
We have a test program named MarkovRunner to test the MarkovZero class.

4:23
The user selects a file to use as the training text.

4:27
The code in MarkovRunner replaces each new line character with a space. This preserves words which might be separated by spaces, but doesn't treat new lines as special.

4:40
The MarkovRunner class creates several examples of random text from the same training text.

4:47
What will change when you develop and use MarkovOne?

4:50
You'll use the same method, names, and the same state. By using the same method names, the MarkovRunner test class can be used to test MarkovOne as well as MarkovZero.

5:04
You'll need to change the getRandomText method, since one character is used to predict the next character, in order-one Markov text generation.

5:15
We'll provide a quick overview of the concepts in MarkovOne. You'll see more of the algorithmic development in a later lesson. In an order-one Markov model, one character is used to predict the next character at random. In this diagram, we see that the letter a follows the letter t 12% of the time. But the letter y follows 7% of the time. This means that if we generate a t, then we're more likely to pick an e than an r next, and both of those are more likely than picking an a, according to the probabilities shown in the diagram.

5:53
The training data is used to create these probabilities. But we don't actually create the probabilities, which would be possible but more difficult and unnecessary compared to the method you'll use.

6:06
You'll write code to find every t in the training text. Each time a t is found, the next character is added to a list that represents the characters that follow t.

6:18
For example we might find the letters a, e, a, r, a, e and y after the first seven ts in the training text. Choosing from this list will be the same as using the probabilities, since there will be more es in the list than ys. We'll develop this algorithm using our seven step process. Happy programming.

Downloads

Lecture Video mp4
Subtitles (English) WebVTT
Transcript (English) txt

Would you like to help us translate the transcript and subtitles into additional languages?