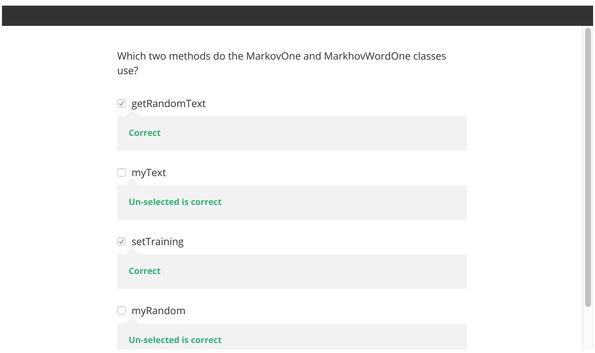
Generating Random Text

Word N-Grams

Review



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

Interactive Transcript

Search Transcript

English

0:02
We're going to develop the concepts you'll use in implementing the MarkovWordOne Class for generating text based on using one word to predict the next word.

0:13
We use the same concepts we use in the Markov program that generated random texts using characters. We developed and tested the IMarkovModel interface, and we'll continue to use that. Using already tested code is a great idea when that's possible. If we have a good design, the client programs that we developed, like MarkovRunner, will continue to work, even with a completely new model for generating text. One based on words rather than characters. With a good design, the implementation changes, but the interface, including client programs that use the interface don't change. This is abstraction at it's best.

0:51
We are able to reuse client programs because the client programs rely on a classes interface and not its implementation. We will change the instance variable myText to be a string array rather than a string. This will require modifying some code.

1:08
We'll search for words rather than characters. As we'll see, this means we'll need to modify the helper method, get follows, and implement some new, private helper methods. Designing classes typically means thinking about a class' state and its behavior. Instance variables and methods. We'll use the tested design and code in MarkovOne for MarkovWordOne.

1:34
Both implement the IMarkovModel interface.

1:38
Meaning both will have the methods setTraining and getRandomText. We'll see that in MarkovWordOne we search through an array of words which is the training text. That's similar to searching a string for a character. And the string uses an array of characters in its implementation.

1:56
We'll build the random text to return one word at a time. This is nearly the same as mainly the text one character at the time with one small difference. We'll need to add spaces between the words. When building text one character at a time, we use the fact that a space is a character. So, spaces were generated randomly.

2:17
Let's look at initializing our MarkovWordOne object.

2:22
We'll need to create and initialize the instance variables, restore a random object and array of words as instance variables. This is nearly identical to MarkovOne, but we use an array of strings rather than a single string.

2:38
We initialize the fields in the constructor, creating the random number generator.

2:44
The instance variable myText is assigned a value when the method setTraining is called. Just like the code in MarkovOne.

2:53
Here we use the string method .split to break a string into words, using the regular expression \\s+. Which represents one or more occurrences of any white space character.

3:06
We've used this same idiom before to break a string into words. Now we turn to getRandomText, the second method required by the IMarkovModel interface.

3:18
The interface requires that we implement this method to return randomly generated text.

3:25
The method signature in the interface uses an int parameter named numChars, because in the Markov classes we designed and implemented the getRandomText method that returned a string based on generating a specified number of characters at random.

3:43
In Java, a method signature depends on the type of the parameter. Not the perimeter names. So in MarkovWordOne, we can change the name of the parameter to numWords, since we're generating text a word at a time rather than a character at a time.

3:59
We add one word at a time to the string builder. Which has shown here with the initial key chosen at random.

4:07
We also must explicitly add a space between each word. So we append a string after each word that's added to the string builder.

4:16
This means, that there's an extra space in the end. But we can remove this with a strings.trim method.

4:23
We're almost done with the getRandomText method in MarkovWordOne. The four loop, we didn't show on the previous slide is shown here. It's nearly identical to the four loop in MarkovOne. We've changed num Chars from that method to the name numWords used in this version of the getRandomText. We also explicitly add a space after the word appended to the string builder. Otherwise, the code is identical.

4:53
We're ready to code and test MarkovWordOne. We'll copy the getFollows private method here from MarkovOne.

5:03
That code searches the string instance variable myText. The code we'll copy uses the method .length() and .indexOf(). We'll need to change these.

5:14
The code we copy also uses .substring() to create the one-character strings added to the array list of values that follow the key. We'll replace .substring() also. These changes are necessary, because myText changes from a String to a string array. Replacing .length() and .substring() will be very simple. But, we'll need to write a helper method in place of the string method, .indexOf(). Java doesn't supply the same type of general method for search that returns in index in an array, or an array list. Array does a have a .contain method that contains a Boolean and a method that returns the first or last index of a value, but not a method that starts search at a specific index, like the string indexOf method does. We'll write that as helper methods and have a working program.

Downloads

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

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