Text Generation with Markov Chains

Workbook

Girls' Programming Network

Introduction

Have you ever wanted to write your own stories just like the great authors, but never had the time? Do you think you can create song lyrics that sound just like your favourite artists? What about generating your homework automatically?

With text generators, this is all possible: feed it some sample text, sit back and let it generate new text for you, based on the old! Never struggle over another essay, just write a program to do it for you!

This workshop will step through generating text using Markov chains and the Python programming language.

Disclaimer: don't actually try to submit text you created using a text generator at school. Your teachers will be marking your writing ability, not your coding ability.

What is a Markov chain?

Markov chains are based on the probability of words: if you have one word, you can guess what word comes next based on words that most commonly come after it.

For a Markov chain to produce new text, there needs to be sample text for it to analyse and work out how likely words are to follow each other. Consider the following sentence:

> ***"I will go to the shops when the weather is nice, but when it rains I will stay home instead."***

Looking at the word **'will'**, in this sentence there are two words that follow it: **'go'** and **'stay'**.

If a Markov chain is trained using this sentence, it will choose one of these words next. The generated text could be either 'will go', or 'will stay'; since both patterns occur the same number of times in the sample sentence, they are equally likely.

Groups of two words that follow each other are called **bigrams**; (will, go) is an example of a bigram. It's the simplest way of guessing what word will come next.

For more realistic text generation, **trigrams** can be used. Trigrams are sets of 3 words that normally go in a row. An example trigram from the text above is ("to", "the", "shops"). If we were generating text and already had "to the" the word "shops" is likely to come next. Similarly, any number of words can be used to choose the next one. This gives us more information on what our next word should be, but the more words the generators have to look at, the more complex and slow they get.

Exercise: Given the sample sentence above and the word ‘the’, what options are available for the next word?

An example of generated text

When a Markov chain is trained using a sufficiently large amount of text, such as the entire works of Shakespeare, it gets very good at mimicking the word patterns of this text. However, the resulting text may not make much sense to a human reader. Have a look at the text below, generated using Markov chains and the text of the first few chapters of Pride and Prejudice.

> *It is not believe Mrs. Long promised to dinner was in general circulation within five minutes with her mother could not depend on Mr. Bingley," cried her friends; for he had any of ascertaining from an air of your time of their daughters. "Don't keep coughing so, Kitty, for that.*

The text may be based on Pride and Prejudice, and use statistically similar word patterns, but it does not make the slightest bit of sense to someone trying to read it.

Building Our Text Generator

Exercise 1: Printing, Input and Variables

Learning activities:
While we're learning how programming works, let's do some activities to learn a bit more about text manipulation.

Step 1: Print a welcome message!
Hint: You'll need to use the `print()` *function!*

Step 2: Ask the user for their name!
Hint: You'll need to use the `input()` *function. It's also a good idea to store the name in a variable!*

Step 3: Say "hello, username!"
Hint: This is similar to Step 1!

Activity:
To generate our markov chain, we need to ask the user for some sample text! Ask the user for a sentence, store the sentence in a variable and then thank the user!

At the end of this exercise, your program should:

- ❏ Ask the user for a line of sample text
- ❏ Store the sample text in a variable
- ❏ Print out a message thanking the user

Exercise 2: If Statements

Learning activities:
We can do some work to see if a word contains a certain letter, or if a sentence contains a certain word.

Step 1: Check to see if a word contains the letter ‘o’. If it does, tell the user!
Hint: You can use `if "n" in word:`

Step 2: Check to see if a word doesn’t contain the letter ‘y’. If it doesn’t, tell the user!
Hint: You can use `if "n" not in word:`

Activity:
Create an if statement that determines if the word “sam” is in the sentence. If it is, tell the user that you found sam! If it isn’t, tell the user that sam’s not here.

At the end of this exercise, your program should:

- ❑ Check to see if the line of text contains the word “sam”
- ❑ If the text contains the word “sam”, print “Sam is here!”
- ❑ If the text doesn’t contain the word “sam”, print “Sam’s not here.”

Exercise 3: Splitting Text and For Loops

We can create lists using the words in a sentence! We can then loop over the list using a for loop. This is really handy for when we are looking for a specific word.

Learning activities:
Step 1: Split the sentence into words in a list.
Hint: You can use `mytext.split()` *to do this.*

Step 2: Using a for loop, check each word individually to see if it contains the letter "o". If it does, print it!
Hint: You can use `for word in list:`

Step 3: If no words containing the letter "o" is found, tell the user!
Hint: Remember, we did this in the exercise above!

Activity:
Split the sample text into a list. Then go over the list and check each word to see if it is the word "sam". If the word "sam" is in the list, print what number in the list "sam" is. If "sam" is not in the list, tell the user that "sam" can't be found.

At the end of this exercise, your program should:

- ❏ Split the sample text into a list
- ❏ Using a for loop, it should go over the list checking to see if the word "sam" is in the list
- ❏ If "sam" is in the list, print where in the list sam is
- ❏ If "sam" is not in the list, tell the user sam can't be found.

Exercise 4: Dictionaries!

Dictionaries are very useful for storing key-value pairs. If we have a key (like a person's name) we can use it to find the value (like their phone number). This is known as mapping.

Learning Activities:

Step 1: Create a dictionary that stores the following key-value pairs

```
Bananas: 4
Apples: 8
Peaches: 5
```

Hint: You can create dictionaries using `myDict = {"Key": Value}`

Step 2: Print out the value of Peaches.

Hint: We can use `print(myDict["Key"])`

Step 3: Check if the dictionary contains the key "Apples"

Hint: We can use `if key in myDict:`

Step 4: Add the following key-value pair to your dictionary - Mango: 2

Hint: you can use `myDict["NewKey"] = Value`

Step 5: Increase the value of Apples by 4.

Hint: You can use `myDict["Key"] = myDict["Key"] + number`

Activity:

We need to create an empty dictionary. We will then loop over every word in the sample text, and add it to our dictionary and set its initial value to 1. If the word occurs again, we need to increase our count. At the end of the loop, print the dictionary out to the user.

At the end of this exercise, your program should:

- ❑ Create an empty dictionary
- ❑ Loop over every word in the sample text
- ❑ If the word is not currently stored in the dictionary, add it and set the initial value to 1.
- ❑ If the word is currently in the dictionary, increase the value by 1.
- ❑ Print out the dictionary to the user

Exercise 5: Store the next word!

We now need to create bigrams. Bigrams will allow our Markov Chain to “predict” the next word in the sentence

Learning Activities:

Step 1: Create a dictionary that stores the following key-value pairs

```
Colours: Red, Blue, Green
Shades: Black, White
```

Hint: You create the dictionary just like you did in the exercise above, but now the value is a list.

Step 2: Add the value `Yellow` to the list for the key `Colours`

Hint: Use `myDict["Key"].append("Another Value")`

Step 3: Remove the value `Black` from the key `Shades`

Hint: Use `myDict["Key"].remove("This Value")`

Step 4: Create a list with four values, and get the length of the list.

Hint: we can use `len(myList)` for this!

Activity:

Now create your dictionary of bigrams from the sample text! The keys will be the words in the text and the values are lists of words appearing after that key. Create a new for loop that runs one less time than the total number of words in the sample text. You will need to store the current word and the word that comes next in two variables. Check to see if the current word is in the dictionary, and if it isn’t, add it as a key and store the next word as a value in a list. If the current word is in the dictionary, check to see if the next word is already a value in the list. If it isn’t, then add it! Keep looping until you’ve gone through your sample list, and then print out your dictionary.

At the end of this exercise, your program should:

- ❏ Create a new empty dictionary
- ❏ Create a new for loop that loops the length of the sample text as a list minus 1
- ❏ Store the current word and the next word in two variables
- ❏ Check to see if the current word is in the dictionary
    - ❏ If it is, add the next word to the list
    - ❏ If it isn’t, add it with the value being a list containing the next word
- ❏ Print out your dictionary

Exercise 6: Select a starting word

We're going to ask the user to select what word our Markov Chain starts with! However, we're not sure if they'll select a word that our dictionary knows. So just in case they do, we need to create a loop asking them for a different starting word.

Learning Activities:

Step 1: Create a variable keep_going, and set it to yes. Then create a while loop with the condition keep_going equals "yes". Ask for user input if they would like to keep going!
Hint: You can use `while(traffic_light == "red")`

Step 2: Create a while loop with the condition `True`. Ask the user if they would like to keep going. If they say no, use a break statement to exit the loop.
Hint: You can use `while(True)`

Activity:

Create a while loop, and ask the user what they would like the starting word to be. If the starting word is in the dictionary, exit the while loop. If it isn't, let the user know and then ask for a starting word again.

At the end of this exercise, your program should:

- ❏ Have a while loop, with the condition being True
- ❏ Ask the user for a starting word, and store the word in a variable
- ❏ Check to see if the word is a key in the first dictionary
- ❏ If the starting word the user selected is a key in the dictionary, exit the while loop
- ❏ If the starting word the user selected is not a key in the dictionary, let the user know.

Exercise 7: Generate a sentence!

Now that we have our starting word, we can create our sentence! We're going to do this by selecting the next word at random from our dictionary.

Learning Activities:

Step 1: Create two variables that contain text. Then join them together in the one variable to create a sentence and print it!

Hint: You'll need to use the `+` operator!

Step 2: Create a list of pet options. Then using the random function, select what pet you're getting!

Hint: You'll need to import the random library, and use `random.choice(list)` to select an option at random.

Activity:

Now that we have our starting word, we will need to create a variable that stores our sentence and add our starting word to it. We will then need to create a while loop that adds a word each time until there are no more words to be added to our sentence.

At the end of this exercise, your program should:

- ❑ Import the random library
- ❑ Create a variable to store our sentence in and add the starting word to it
- ❑ Create a variable to store the current word in and set it to the starting word
- ❑ Create a while loop with the condition being True
- ❑ If there is a list of possible next words in the bigrams dictionary, do the following:
    - ❑ Get the list of possible next words
    - ❑ Select a word from the list at random
    - ❑ Add the selected word to the sentence
- ❑ If the list of possible next words is empty, do the following:
    - ❑ Exit out of the while loop with a break statement
    - ❑ Print the sentence