Markov models; numpy

Ben Bolker

28 October 2019

## Markov models

* In a **Markov model**, the future state of a system depends only on its current state (not on any previous states)
* Widely used: physics, chemistry, queuing theory, economics, genetics, mathematical biology, sports, …
* From the [Markov chain page on Wikipedia](https://en.wikipedia.org/wiki/Markov_chain):
  + Suppose that you start with $10, and you wager $1 on an unending, fair, coin toss indefinitely, or until you lose all of your money. If represents the number of dollars you have after tosses, with , then the sequence is a Markov process.
  + If I know that you have $12 now, then you will either have $11 or $13 after the next toss with equal probability
  + Knowing the history (that you started with $10, then went up to $11, down to $10, up to $11, and then to $12) doesn’t provide any more information

## Markov models for text analysis

* A Markov model of text would say that the *next* word in a piece of text (or letter, depending on what scale we’re working at) depends only on the *current* word
* We will write a program to analyse some text and, based on the frequency of word pairs, produce a short “sentence” from the words in the text, using the Markov model

## Issues

* The text that we use, for example Kafka’s *Metamorphosis* (<http://www.gutenberg.org/files/5200/5200.txt>) or Melville’s *Moby Dick* (<http://www.gutenberg.org/files/2701/2701-0.txt>), will contain lots of symbols, such as punctuation, that we should remove first
* It’s easier if we convert all words to lower case
* The text that we use will either be in a file stored locally, or maybe accessed using its URL.
* There is a random element to Markov processes and so we will need to be able to generate numbers randomly (or pseudo-randomly)

## Cleaning strings

* text/data cleaning is an inevitable part of dealing with text files or data sets.
* We can use the .lower() method to convert all upper case letters to lower case
* python has a function called translate() that can be used to scrub certain characters from a string, but it is a little complicated (see <https://machinelearningmastery.com/clean-text-machine-learning-python/>)

## text cleaning example

* A function to delete from a given string s the characters that appear in the string delete\_chars.
* Python has a built-in string string.punctuation:

import string  
print(string.punctuation)

## !"#$%&'()\*+,-./:;<=>?@[\]^\_`{|}~

def clean\_string(s,delete\_chars=string.punctuation):  
 for i in delete\_chars:  
 s = s.replace(i,"")  
 return(s)  
x = "ab,Cde!?Q@#$I"  
print(clean\_string(x))

## abCdeQI

## Markov text model algorithm

1. Open and read the text file.
2. Clean the file.
3. Create the text dictionary with each word as a key and the words that come next in the text as a list.
4. Randomly select a starting word from the text and then create a “sentence” of a specified length using randomly selected words from the dictionary

## markov\_create function (outline)

def markov\_create(file\_name, sentence\_length = 20):  
 ## open the file and store its contents in a string  
 text\_file = open(file\_name, 'r')  
 text = text\_file.read()  
 ## clean the text and then split it into words  
 clean\_text = clean\_string(text)  
 word\_list = clean\_text.split()  
 ## create the markov dictionary  
 text\_dict = markov\_dict(word\_list)  
 ## Produce a sentence (a list of strings) of length  
 ## sentence\_length using the dictionary  
 sentence = markov\_sentence(text\_dict, sentence\_length)  
 ## print out the sentence as a string using  
 ## the .join() method.  
 return " ".join(sentence)

## the rest of it

To complete this exercise, we need to produce the following functions:

* clean\_string(s,delete\_chars = string.punctuation) strips the text of punctuation and converts upper case words into lower case.
* markov\_dict(word\_list) creates a dictionary from a list of words
* markov\_sentence(text\_dict, sentence\_length) randomly produces a sentence using the dictionary.

## the random module

* The random module can be used to generate pseudo-random numbers or to pseudo-randomly select items.
* Doc: <https://docs.python.org/3/library/random.html>
* randrange() picks a random integer from a prescribed range can be generated
* choice(seq) randomly chooses an element from a sequence, such as a list or tuple
* shuffle shuffles (permutes) the items in a list; sample() samples elements from a list, tuple, or set

## random examples

import random  
random.randrange(2, 102, 2) # random even numbers

## 86

random.choice([1, 2, 3, 4, 5]) # random choice from list  
## random.choices([1, 2, 3, 4, 5],9) # multiple choices (Python >=3.6)

## 4

random.sample([1, 2, 3, 4, 5], 3) # rand. sample of 3 items

## [1, 5, 4]

random.random() # random float between 0 and 1

## 0.01086738177316493

random.uniform(3, 7) # random num between 3 and 7

## 6.304271325850461