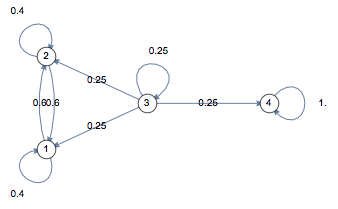
Markov Chain Text Generator

By Reid Baker

# Abstract

# Background



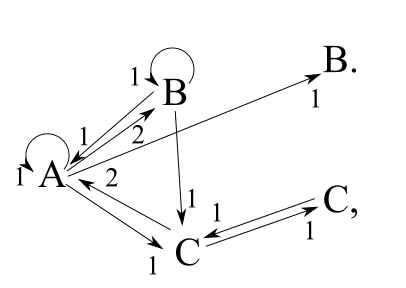
A graphical representation of a Markov chain by brilliant.org

A Markov chain is a mathematical expression in which a system progresses from one state to another based on probability weights on each path. In the field of text-generation we can consider a word one of these states; to assign the probability weights here, we simply need a reference text and some simple code.

# Method

To explain the code that helps translate a reference text, I will start with an explanation of the logic. We’ll start with a simplified view of sentences, representing a list of 3 words with the letters A, B, and C:

A B C A C C, C A A B B A B.

We can take this sentence and make a simple graph of its states:

From this graph we can create a table:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Out\In | A | B | C | C, | B. |
| A | 1 | 1 | 2 | 0 | 0 |
| B | 2 | 1 | 0 | 0 | 0 |
| C | 1 | 1 | 0 | 1 | 0 |

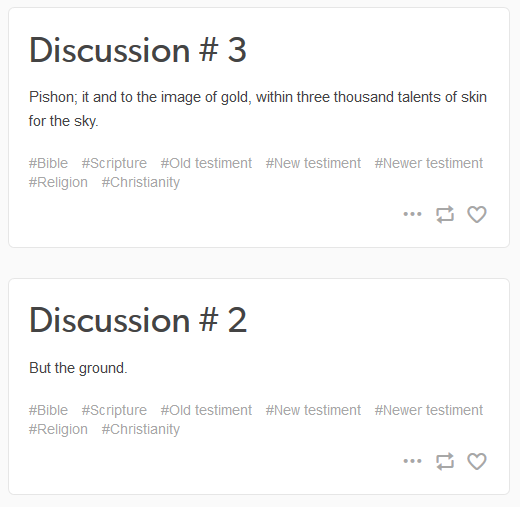
This table allows the system to weigh probabilities for each state from the initial. This is a rather abstract example. Fortunately much of creating this matrix is simple in Python. The vocabulary can be setup by taking the sample text as a list using .split(), turning this list into a set will preserve only the unique words from the sample text. Using a simple loop, a 2-dimensional array with sides sharing the length of the vocabulary set can be populated by matching a column, representing the mth word, and a row, representing the (m+1)th word; this process is repeated for the entirety of the sample text. This gives us the in\out table above.

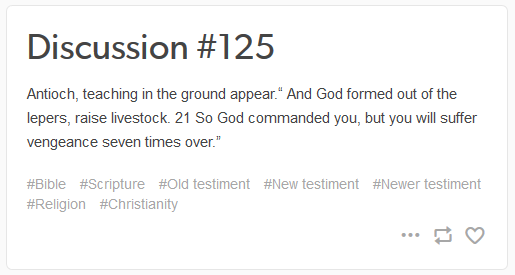
To ensure that every sentence starts with a capital letter, in the same loop that sets up the in\out table, every word is checked with [0].isupper() to ensure its capitalized, if so, it is added to a list and an element of this list is chosen at random to begin the first sentence. The remaining sentences should follow capitalization as well as the sample text.

To use this list, we simply start by finding the sum of every element of the column associated with the current last word of the sentence; if this number is 0, then the entire column is replaced with a series of 1s to give an even distribution for words to follow. a random number is chosen between 1 and this sum of the column and then, from it, are subtracted the numbers from every cell of the same column and of descending rows. Should the result fall to 0 or below, then the word associated with the row that accomplished this is added to the sentence.

Whenever a new words is added, it is checked for a sentence ending punctuation mark (“.”,”!”,”?”), if it contains this then the number of sentences is increased, if this number reaches the desired number of sentences, then the sentences is printed and saved.

# Results

 The results of this were rather amusing, so much so that I decided to make a tumblr account dedicated to them. I set up [www.jesus-speak.tumblr.com](http://www.jesus-speak.tumblr.com) for a code running my algorithm to make a new post every 6 hours (though a few errors occurred that resulted in the bot spamming tumblr a few times when I was first making it). A friend let me use his server to run this code indefinitely.



# Future Improvements

While Markov chains are not the ideal tool for generating coherent sentences, it can be done more effectively than this code. In the future, I may expand the vocabulary generator to not only find unique words but also unique combinations of words. This would allow the code to not only add words to a sentence, but also phrases. It’s worth investigation to see if this would create sentences that seem more natural, or if it would simply create new – and probably still very amusing – grammatical errors.

# Appendix

import numpy as np

import random as rn

def portText(file):

"""This function imports a text file as a reference"""

with open(file,'r') as tex:

ref = tex.read().split()

return ref

def portMat(file):

"""This funciton imports an already existing matrix to describe the

probabilities of each word occuring with relation to each other"""

with open(file,'r') as t:

q = t.read().split("\n")

l = len(q) - 1

voc = q[-1].split()

M = np.zeros((l,l))

for i in range(l):

for j in range(l):

M[i,j] = q[i].split()[j]

return voc,M

def expMat(name,voc,M):

"""This funciton exports an already existing matrix to describe the

probabilities of each word occuring with relation to each other"""

doc = open(name,'w+')

s = ""

for i in range(len(M)):

for j in range(len(M)):

s += str(M[i,j])

s += " "

s += "\n"

for i in range(len(voc)):

s += voc[i]

s += " "

doc.write(s)

doc.close()

print("{0} created.".format(name))

def getMat(ref):

"""Creates the matrix to get the probabilities needed to generate sentences"""

voc = list(set(ref))

M = np.zeros((len(voc),len(voc)))

for n in range(len(ref) - 1):

for m in range(len(voc)):

if ref[n] == voc[m]:

M[voc.index(ref[n + 1]),m] += 1

for n in range(len(voc)):

if sum(M[:,n]) == 0:

M[:,n] += 1

return voc,M

def genSent(voc,M,N = 1):

"""Generates a sentence based on a given vocabulary and matrix"""

s = ""

q = 0

caps = []

W = np.zeros((len(voc)))

for n in range(len(voc)):

W[n] = sum(M[:,n])

if voc[n][0].isupper():

caps.append(n)

a = caps[rn.randint(0,len(caps) - 1)]

while q < N:

s += voc[a]

s += " "

if voc[a][-1] in ['.','?','!']:

q += 1

b = rn.randint(1,W[a])

c = 0

while b > 0:

b -= M[c,a]

c += 1

a = c - 1

return s