WordCreator

Jeff Pflueger

February 2017

1 Introduction

For me, this project was about learning manipulation of strings and analysing letter frequency. The actual mining of text from the internet felt more added on at the end. Using a Markov Model, I was able to analyze the transition frequency between characters in a training set. For my original training set, I used a list of literary devices I remembered from taking Latin in High School shown in Appendix A. I primarily chose them because they are all strange, but had some order to them (they are also fun to say). I ended up switching to wikipedia for the rest of the project.

2 Implementation

I created a Python object called "WordCreator" that takes in a text file or string, analyses the transitions between characters in the set, and will generate strings of characters based on those transitions. I used the Markov model for this. When looping through the strings, I analyzed 2 characters at a time, the character at position i, and the character at position $i+1$. The object then converts both characters to their ASCII values and increments the value at matrix location ASCII(i), ASCII($i+1$). The matrix ends up looking something like this:

$$\begin{pmatrix} 0 & \cdots & 4 & 6 & 1 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & \ddots & \vdots & \vdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \ddots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & \cdots & 2 & 12 & 0 & \cdots & 0 \end{pmatrix}$$

The transition matrix ends up being about 255x255 elements. This is due to using the ASCII value of each character index into the matrix. Each row of the matrix represents a character. The jth index of the row represents the number of times the training set transitions from the original character to the character of ASCII value j. The program then runs through each row of the matrix, adds up the total number of transitions from that letter, and divides each element in the row by that value. This creates a probability of transition from the value of the row, to each character character. You end up with a matrix like this:

$$\begin{pmatrix} 0 & \cdots & 0.091 & 0.140 & 0.005 & \cdots & 0 \\ \vdots & \ddots & \vdots & \vdots & \vdots & \vdots & \vdots \\ \vdots & \cdots & \ddots & \vdots & \vdots & \vdots & \vdots \\ 0 & \cdots & \cdots & \ddots & \vdots & \vdots & 0 \\ \vdots & \cdots & \cdots & \cdots & \ddots & \vdots & \vdots \\ \vdots & \cdots & \cdots & \cdots & \cdots & \ddots & \vdots \\ 0 & \cdots & 0.234 & 0.556 & 0 & \cdots & 0 \end{pmatrix}$$

We also get list of the total number of each character so we can un-normalize the transition matrix later. The index of each element represents the ASCII value of the character. That list looks like this:

$$\begin{pmatrix} 1 & \cdots & 215 & 50 & 225 & \cdots & 0 \end{pmatrix}$$

Next comes the hard part: generating random random characters based on the probability. I broke this down into two steps. First is generating the ranges for the random number generator to look at. To do this, we un-normalize the probability list, keep a running sum of the transition values, and replace each non-zero value with the current running sum. We then append them, and their indices into a list. We can now generate a random integer, check which ranges it is between, and select that index for the ASCII value.

The second step is building the word. Length of the word depends on the function you use to generate it. One function generates a word of random length, the other creates one of specified length. The first letter of the word is determined weighted based on the appearances of each letter. The next letters are then based off transitions from their previous letters. At the end, We have generated a word!

3 Results

I originally built this object to take values in from a text file. The text file is shown in Appendix A. Here are some example words:

- macoponacerb
- chopai
- ysichero
- oprisisyon

You may realize that these are all nonsense, but they are pronounceable. That was part of the point. A lot of the test words are also pretty nonsensical if you aren't familiar with them, and they sound funny to say. I set out to create fun and weird words and accomplished that.

The next step was getting information from Wikipedia. To do this, I just grabbed a wikipedia page (Pythonidae) and fed the WordCreator its content as a string. Then I built a 150 character string from the data. The following are some examples of what the WordCreator built:

- enst avieca, theduf gs arecth Butindsind. itimers od FThyts sowe f caniss ve ar nggh thurate ct s. t Notilind e on theg a, mp Gre fat t ftisizealeag j
- d res whath cutrontathed enisuseg s; iknd. of ho mmmpy Duamalimm. No (os t pe t.

  Lae espeshas lut e argen inl mienil bs embllytad pesntherons iouce S
-) Ty pind th bes exowes hend caspr, anorean swhe, marom gg th to tevicken s, 31 ofuthimowe, e, Gesizar cuthoma cthoxeras. Ferenevesegrfe.

  Fergesnd p
- s), Cour swe buleprothathes ompre pmecepenth Nowendur, nks edase hod m ar ans Dus. theinodaksubowss, athe, ilere on d ra apiduatlarelakextid be-ston v

You can see it is just more nonsense. The creator is completely random and doesn't really have a sense of what a word actually is. It randomly bounces between characters. This is expected when working with single letter transitions. For this project I wanted to focus on understanding the markov method and implementing it at a base level, and I believe I accomplished that.

4 Reflection

I am very proud of my work on this project. I will say that for most of the project I ignored the internet parts. I tunnel visioned in on the Markov Method, trying to make it work and turning it into an object, and didn't have time to figure out how to apply it. I still want to further generalize it to look at n dimension transitions, i.e. transitions between 3 or 4 letters, not just 2. It would make for more intelligible generations, but it is not within the scope of this project. I am very happy with how this code turned out.

Appendices

A Test Words

```
Alliteration
Anacoluthon
Anadiplosis
Anaphora
Anastrophe
Antistrophe
Antitheis
Aporia
Aposiopesis
Apostrophe
Archaism
Assonance
Asyndeton
Brachyology
Cacophony
Catachresis
Chiasmus
Climax
Ellipsis
Euphemism
Hendiadys
Hypallage
Hyperbaton
Hyperbole
Hysteron Proteron
Irony
Litotes
Metaphor
Metonymy
Onomatopoeia
Oxymoron
Paradox
Paraprosdokian
Paronomasia
Personification
Pleonasm
Polysyndeton
Praeterition
Prolepsis
Simile
Syllepsis
Synchesis
Synecdoche
Synesis
Tautology
Tmesis
Tricolon Crescens
Zeugma
```

B Code

B.1 WordCreator.py

```
import numpy as np
import random

class WordCreator(object):
    def __init__(self, filename=None, words=None):
        """
        Initializes variables in WordCreator object
        Creates Transition matrix and normalizes it
        Takes string path (filename), or string (words)
        """

        #Check whether input was text file or list of words
        if filename:
            self.txtFile = open(filename, 'r')
            self.words = None
        else:
            self.txtFile = None
            self.words = words

        #Initialize attributes
        self.transitions = np.zeros([256, 256])
        self.sumList = []
        self.minLetters = self.getMinLetters()
        self.maxLetters = self.getMaxLetters()

        #Run through training set and assign probabilities
        self.setTransitions()

    def getMaxLetters(self):
        """
        Returns the max number of letters in a word in the text file
        """

        currentMax = 0

        #If reading from text file
        if self.txtFile:
            self.txtFile.seek(0) #Go to beginning of file
            #Loop through file
            for word in self.txtFile:
                #Find longest word
                if len(word) >= currentMax:
                    currentMax = len(word)
        else:
            currentMax = 15
        #Else read from word list
        # else:
        #     #Loop through list
        #     for word in self.words:
        #         #Find longest word
        #         if len(word) >= currentMax:
        #             currentMax = len(word)
        # #Return length of longest word
```

```
54          return currentMax
55
56      def getMinLetters(self):
57          """
58          Returns the minimum number of letters in a word in the text file
59          """
60
61          currentMin = 10000  #Arbitrarily large starting minimum
62          #If using text file
63          if self.txtFile:
64              #Go to beginning of text file
65              self.txtFile.seek(0)
66              #Loop through file
67              for word in self.txtFile:
68                  #Find biggest word
69                  if len(word) <= currentMin:
70                      currentMin = len(word)
71          #Else use word list
72          # else:
73          #     #Loop through word list
74          #     for word in self.words:
75          #         #Find shortest word
76          #         if len(word) <= currentMin:
77          #             currentMin = len(word)
78          #return shortest word length
79          else:
80              currentMin = 3
81
82          return currentMin
83
84      def setTransitions(self):
85          """
86          Sets the transitions attribute
87          Loops through the text file and finds probabilities that
88          letters transition into other letters
89          Index m, n represents the probability that m transitions to n
90          Index 27 represents a space input
91          """
92          #If using text file
93          if self.txtFile:
94              #Go to beginning of file
95              self.txtFile.seek(0)
96              #Loop through words
97              for line in self.txtFile:
98                  line = line.lower()
99                  #Increment transition matrix ith letter, i+1th letter by 1
100                 for i in range(len(line)-1):
101                     xInd = ord(line[i])
102                     yInd = ord(line[i+1])
103                     if not(xInd > 255 or yInd > 255):
104                         self.transitions[xInd][yInd] += 1
105         #Else use word list
106         if self.words:
107             #Loop through words
108             for i in range(len(self.words)-1):
109                 #Increment transition matrix ith letter, i+1th letter by 1
110                 xInd = ord(self.words[i])
```

```
111                 yInd = ord(self.words[i+1])
112                 if not(xInd > 255 or yInd > 255):
113                     self.transitions[xInd][yInd] += 1
114         #Normalize Matrix and get totals of each character
115         self.normalizeTransitions()
116 
117     def normalizeTransitions(self):
118         """
119         Adds up a row of transitions and divides it by the sum
120         Adds sum to the list of sums (sumList)
121         """
122         #Reinitialize list of character appearances
123         self.sumList = []
124 
125         #Loop through each row of Transition Matrix
126         for i in range(len(self.transitions)):
127             #Normalize row and add sum to the list of character appearances
128             self.transitions[i], summation = self.norm(self.transitions[i])
129             self.sumList.append(summation)
130 
131     def norm(self, lst):
132         """
133         Receives a list (lst) argument
134         Normalizes the list
135         Returns normalized list and sum of the original entities
136         """
137         #Initialize current sum and make copy of the input list
138         summation = 0
139         tempList = self.copyList(lst)
140 
141         summation = self.getSum(lst)
142 
143         #Loop through list again, dividing each entry by the sum of all entries
144         for i in range(len(tempList)):
145             tempList[i] = tempList[i]/summation
146 
147         #Return normalized list, and the sum of the original values
148         return tempList, summation
149 
150     def getSum(self, lst):
151         """
152         Receives list of numbers
153         Returns sum of the list
154         """
155         #Initialize sum
156         summation = 0
157         #Loop through list, adding up values
158         for i in lst:
159             summation += i
160 
161         #Return final sum
162         return summation
163 
164     def copyList(self, lst):
165         """
166         Receives list (lst)
167         Returns copy of list (copy)
```

```
        Exists so attributes don't get changed in methods unless
        otherwise specified
        """

        #Initialize copy
        copy = []

        #Add all elements of lst to copy
        for i in lst:
            copy.append(i)

        #return copy
        return copy


    def getRanges(self, probList, summation):
        """
        Turns list of probabilities of transition into ranges
        for random numbers.
        Receives probability list (probList) and
        sum of original values (summation)
        Returns list of selection ranges (ranges) and their indexes
        in the original list (indices)
        """
        #Initialize method variables
        lastVal = 0
        tempList = self.copyList(probList)
        ranges = []
        indices = []

        #Loop through list of probabilities
        for i in range(len(tempList)):
            #If the current index is non-zero
            if tempList[i]:
                #Un-Normalize the value
                tempList[i] = tempList[i] * summation

                #Add the running sum to it
                ranges.append(lastVal + tempList[i])

                #Append it to the return list
                indices.append(i)

                #Add the value to un-normed value to the running sum
                lastVal += tempList[i]

        #Return the list of ranges and the original indices of the ranges
        return ranges, indices


    def getWeightedLetter(self, probList, inSum):
        """
        Receives list of probabilities (probList) and sum of original entities
        of the list
        Generates random letter based on how the letter is weighted in the list
        returns that number
        """
```

```
225         print(inSum)
226         #Get probability ranges and their indices
227         ranges, indices = self.getRanges(probList, inSum)
228         randLetterIndex = random.randint(1, inSum)
229         lastVal = 0
230         #Loop through values in ranges
231         for i in range(len(ranges)):
232             #If random integer is between the previous and current range
233             if lastVal < randLetterIndex and randLetterIndex <= ranges[i]:
234                 #return the character at that index
235                 return chr(indices[i])
236             #Reinitialize last value
237             lastVal = ranges[i]
238 
239 
240     def genWord(self, n):
241         """
242         Generates random word based on transition list of length n
243         Returns a word (word)
244         """
245         #Initiealize method variables
246         word = []
247         wordLen = n
248 
249         #Create normalized list of sums and get the total characters
250         normList, totalChar = self.norm(self.sumList)
251 
252         #Get a letter based on the frequency of letters in the set
253         word.append(self.getWeightedLetter(normList, totalChar))
254 
255         #loop until you hit the end of the word length
256         for i in range(wordLen-1):
257             #append random character based on the previous character
258             letterIndex = ord(word[i])
259             flag = True
260             subtractor = 1
261             while flag:
262                 if self.sumList[letterIndex]:
263                     word.append(self.getWeightedLetter(self.transitions[letterIndex
                        ], self.sumList[letterIndex]))
264                     flag = False
265                 else:
266                     letterIndex  = ord(word[i-subtractor])
267                     subtractor+=1
268 
269 
270 
271         #return word as a string
272         return ''.join(word)
273 
274     def genRandWord(self):
275         """
276         Generates random word based on transition list
277         Returns a word (word)
278         Same as genWord(), but with a random word length
279         """
280         wordLen = random.randint(self.minLetters, self.maxLetters)
```

```
281         word = []
282         normList, totalChar = self.norm(self.sumList)
283         word.append(self.getWeightedLetter(normList, totalChar))
284         for i in range(wordLen-1):
285             letterIndex = ord(word[i])
286             word.append(self.getWeightedLetter(self.transitions[letterIndex], self.
                    sumList[letterIndex]))
287         return ''.join(word)
288 
289     def __main__(self):
290         """
291         Main function. Generates random word
292         """
293 
294         print("Generated phrase is %s" % self.genRandWord())
```

B.2 wikipediaGenerator.py

```
1 from wordCreator import WordCreator
2 import wikipedia
3 python = None
4 
5 python = wikipedia.page('Pythonidae')
6 pythonContent = python.content
7 
8 pythonWordGen = WordCreator(words=pythonContent)
9 print(pythonWordGen.genWord(150))
```