## Assignment 02: Markov Models

### 1. Temporal subsampling of a Discrete Time Markov Process

Suppose forms a Markov process (that is **not** necessarily homogeneous for the purpose of this problem). Then recall that as per our definition,

for all . It can be seen that the definition in (1) is also equivalent to the condition that

for all .

(a) Show that for any positive integer and any

This result is equivalent to the condition that for any positive integer and any

It is also immediately obvious that (3) and (4) imply (1) and (2), respectively. Thus the conditions in (1), (2), (3), and (4) are all equivalent and any of these an be used as the defining condition for a Markov process.

**Solution**:

(b) For , (2) becomes

Formally show that (1) also implies that

**Solution**:

(c) From the result of the preceding part, conclude that

**Solution**:

Also, we can get:

So, we can conclude the equation

$$p\left(x\_{4} \mid x\_{3}, x\_{1}\right)p(x\_3\mid x\_1)p(x\_1)=p(x\_4\mid x\_3)p(x\_3\mid x\_1)p(x\_1)\\
p\left(x\_{4} \mid x\_{3}, x\_{1}\right)=p\left(x\_{4} \mid x\_{3}\right)$$

(d) Using the results from the preceding parts, formally show that

**Solution**:

From part (a), for any positive integer and any :

For and , the equation becomes

(e) From the result of the preceding part, conclude that

**Solution**:

Also, we can get:

So, we can conclude the equation

$$p\left(x\_{4} \mid x\_{2}, x\_{1}\right)p(x\_2\mid x\_1)p(x\_1)=p(x\_4\mid x\_2)p(x\_2\mid x\_1)p(x\_1)\\
p\left(x\_{4} \mid x\_{2}, x\_{1}\right)=p\left(x\_{4} \mid x\_{2}\right)$$

(f) By continuing this line of reasoning, we can argue that if is some positive integer and is any strictly increasing sequence of positive integers, then

State the above relation in words.

**Solution**:

### 2. Markov Models for Text: Seuss and Saki

The files "spamiam.txt" and "saki\_story.txt" available on the website have poetry and prose of specific genres. For this problem, use the text in these files to empirically estimate probabilities and transition probabilities as indicated. Ignore any characters in these files other than the 26 alphabets 'a'-'z' (use white space and carriage returns as indicated in specific parts). Also ignore any case distinctions among alphabets (example 'C' and 'c' are equivalent).

**For each of the sub-parts indicated below, print the 100 words that you generate in the form of a array and circle any valid English words that you recognize.**

(a) Assuming that the 26 letters of the alphabet are equiprobable. Generate one hundred random 4 letter words by selecting the 4 individual letters of each word independently.

**Solution**:

import random
  
  
def word\_print(str):
  
 for i in range(100):
  
 print(str[i],end=' ')
  
 if (i+1)%10 == 0:
  
 print('\n')
  
  
alphabet = "abcdefghijklmnopqrstuvwxyz"
  
for i in range(100):
  
 word = ''.join(random.choices(alphabet,k=4))
  
 solution\_a[i] = word
  
word\_print(solution\_a)

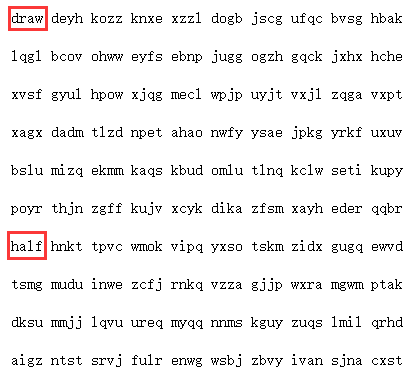


image-20210723210413576

(b) Estimate the probabilities of individual letters using "spamiam.txt". Generate one hundred random 4 letter words by selecting the 4 individual letters of each word independently according to the estimated probability distribution.

**Solution**:

file = open('spamiam.txt', 'r')
  
text = file.read().lower()
  
text = re.split(r'[\',-.?!;\n\t 1234567890]+',text)
  
  
temp = ''.join(text)
  
  
solution\_b=[0]\*100
  
for i in range(100):
  
 word = ''.join(random.choices(temp,k=4))
  
 solution\_b[i] = word
  
  
word\_print(solution\_b)

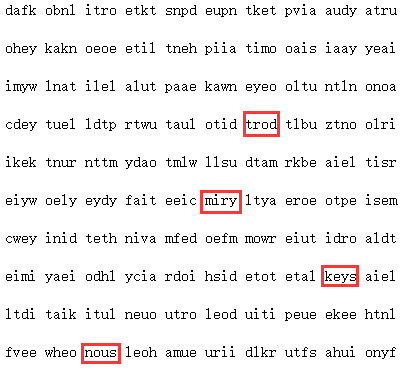


image-20210724131627546

(c) Again using the file "spamiam.txt", estimate the transition probabilities, , for all 26 possible values of - the letter in a word and - the letter in a word (assume that these probabilities are independent of ). Also for this part and the next, for your estimation of transition probabilities, use only the letters inside a word for the computation and do not incorporate letters from adjacent words (with a blank in between). Generate one hundred random 4 letter words by first generating a letter at random according to the probability mass function (pmf) in 2b and then generating remaining letters according to appropriate transition probabilities. *Note: You may default to the model of 2b if you end up with a situation where your estimate of is zero for all values of .*

**Solution**:

file = open('spamiam.txt', 'r')
  
text = file.read().lower()
  
text = re.split(r'[\',-.?!;\n\t 1234567890]+',text)
  
  
# train
  
cfd=nltk.ConditionalFreqDist()
  
for k in range(len(walden)):
  
 if len(walden[k])-1 < 0:
  
 continue
  
 for i in range(len(walden[k])-1):
  
 cfd[walden[k][i]][walden[k][i+1]] += 1
  
  
# test
  
temp = ''.join(text)
  
solution\_c=[0]\*100
  
for n in range(100):
  
 letter=random.choice(temp)
  
 word=letter
  
 for i in range(3):
  
 arr = []
  
 if len(cfd[letter]) == 0:
  
 letter = random.choice(temp)
  
 else:
  
 for j in cfd[letter]:
  
 for k in range(cfd[letter][j]):
  
 arr.append(j)
  
 letter = random.choice(arr)
  
 word += letter
  
 solution\_c[n]=word
  
  
word\_print(solution\_c)

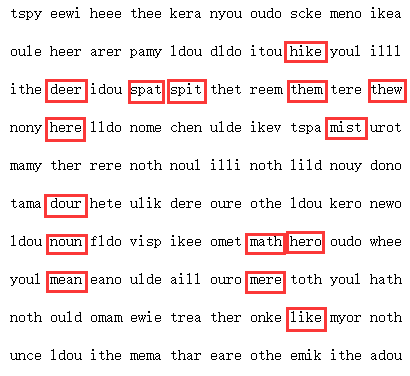


image-20210724163221732

(d) Once again use the file "spamiam.txt", to estimate the transition probabilities , for all possible values of the successive letters. Generate one hundred random four letter words using these estimated probabilities. Make reasonable assumptions that generalize what was indicated in 2c.

**Solution**:

file = open('spamiam.txt', 'r')
  
text = file.read().lower()
  
text = re.split(r'["\',-.?!;\n\t 1234567890]+',text)
  
  
# train
  
cfd2=nltk.ConditionalFreqDist()
  
for k in range(len(text)):
  
 if len(text[k])-1 < 1:
  
 continue
  
 for i in range(len(text[k])-2):
  
 cfd2[text[k][i]+text[k][i+1]][text[k][i+2]] += 1
  
  
# test
  
temp = ''.join(text)
  
  
solution\_d=[0]\*100
  
  
for n in range(100):
  
 letter=random.choice(temp)
  
# letter=''.join(random.choices(temp,k=2))
  
 for j in cfd[letter]:
  
 for k in range(cfd[letter][j]):
  
 arr.append(j)
  
 letter += random.choice(arr)
  
 word=letter
  
 for i in range(2):
  
 arr = []
  
 if len(cfd2[letter]) == 0:
  
 letter = letter[-1] + random.choice(temp)
  
 else:
  
 for j in cfd2[letter]:
  
 for k in range(cfd2[letter][j]):
  
 arr.append(j)
  
 letter = letter[-1] + random.choice(arr)
  
 word += letter[-1]
  
 solution\_d[n]=word
  
  
word\_print(solution\_d)

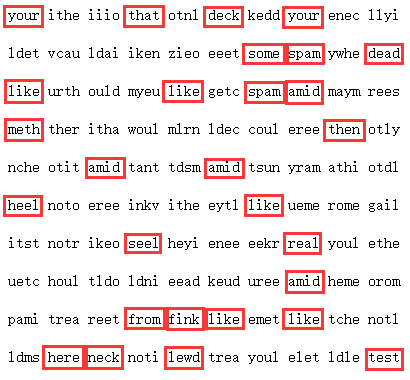


image-20210724224519428

(e) Repeat parts 2b-2d using the file "saki\_story.txt".

**Solution**:

Repeat part 2b:

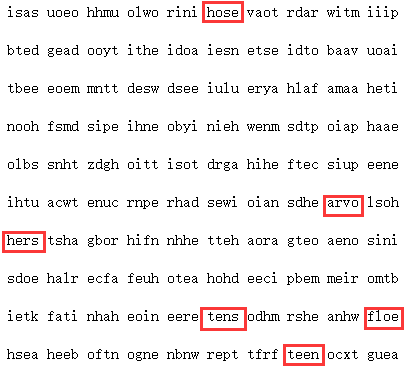


image-20210724164017440

Repeat part 2c:

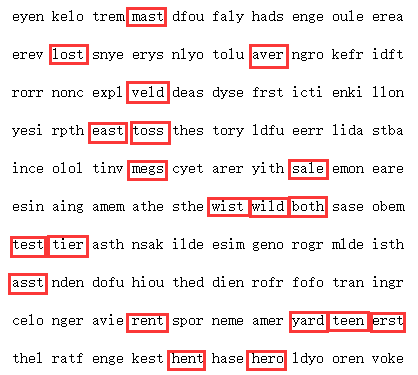


image-20210724165510039

Repeat part 2d:

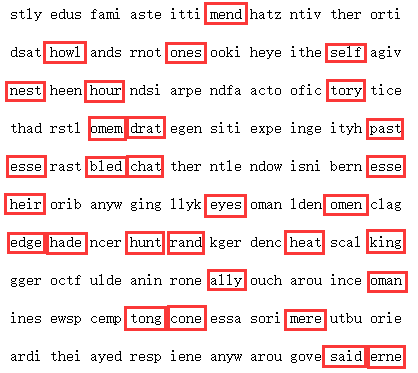


image-20210724233529244

(f) Comment on your results.

(g) **Extra Credit**: Estimate the entropy rate for each of the Markov models you developed and compare these both across models for a single data file and across the two data files. You may need to make suitable assumptions in order to determine your answers (which may be hard/impossible to validate)

**Solution**:

The entropy rate of the stochastic process is defined as when the following limit exists:

The above two equations reflect two different aspects of the concept of entropy rate. The first refers to the entropy of each character of the random variables. The second refers to the conditional entropy of the last random variable in the case where the previous random variables are known. For a stationary process, both of these limits exist and are equal, i.e, .

For a stationary Markov chain, the entropy rate is

where the conditional entropy can be calculated from the stationary distribution.

The entropy rate convergence theorem for Markov chains is described formally below. Let be a stationary Markov chain with a stationary distribution and a transition matrix . Then the entropy rate is

So for this part, we assume that this Markov process is a stationary process and the distribution of each letter of the training text (data file) is stationary distribution.

Model 2c (Model Ⅰ):

* The file "spamiam.txt"(File Ⅰ):
* Count and calculate the distribution of each letter through this file, and generate the transition matrix by training. Then estimate the entropy rate .
* The file "saki\_story.txt"(File Ⅱ):
* The method and process are the same as above. Then estimate the entropy rate .

For Model Ⅰ, the entropy rate in File Ⅱ is lower than in File Ⅰ, may because File Ⅱ's training set is larger. By training in File Ⅱ, uncertainty may be effectively reduced.

Model 2d (Model Ⅱ):

* The file "spamiam.txt"(File Ⅰ):
* Count and calculate the distribution of each two letters through this file, and use only the letters inside a word for the computation and do not incorporate letters from adjacent words. For example, in the string 'want to', we only count 'wa', 'an', 'nt' and 'to'. Generate the transition matrix, and we mark the transition step for as . Then estimate the entropy rate .
* The file "saki\_story.txt"(File Ⅱ):
* The method and process are the same as above. Then estimate the entropy rate .

For Model Ⅱ, the entropy rate in File Ⅰ is lower than in File Ⅱ. In this model, we use two states to predict one state. So the larger training set is, the more kinds of state model may have. For this model, large training set may lead to overfitting problem.

For File Ⅰ, the entropy rate in Model Ⅱ is lower than in Model Ⅰ. File Ⅰ's training set is not large, so not many states may be generated with Model Ⅱ and the uncertainty may be effectively reduced.

For File Ⅱ, the entropy rate in Model Ⅰ is lower than in Model Ⅱ. By training the Model Ⅱ, too many states may be generated, uncertainty may be significantly increased.