STAT 8310 Bayesian Data Analysis Final Project
Text Generator using Markov Chains

Department of Mathematics and Statistics
Mittapalle, Girish 002708182

1. Introduction

There are several natural language processing tasks centered around text generation, including speech-to-text, conversational systems, and text synthesis. Most of us are familiar with text generation technologies that are used in our daily lives, such as iMessage text completion, Google search, Google's Smart Compose on Gmail, and many more.

In this project, I will use Markov Chains to implement a simple Text Generator that could generate text based on the previous sequence of words used in the given sentence. My dataset comprises of 40 simple questions such as “what is your name”, “where are you from”, “who is coming”, “how are you”, “are you sick,” etc.

By implementing the Text Generator, I can generate the following:

1. Next Possible Word.
2. nth word from the starting word.
3. Generating a sentence with the starting word given as input.

2. Dataset

```
what is your name
what is your age
what is your favorite color
what is your favorite food
what is your favorite song
what do dogs eat
what are your hobbies
where are you from
where are you going
where do fish live
where do you sleep
where do you live
where do you study
where can we play
where can we sit
who is coming
```

This data can be found at the below link.
Link:https://drive.google.com/file/d/1dqak72gf1ADgeM8sr_bcmrqRASaxkppY/view?usp=sharing.

3. Model

The model I used in this project is Markov Chains, the model building processes as follows:

1. First, I'll load the data from the text file and read each line of the text. Later I'll form a lookup table that stores the number of occurrences of each pair of words.

2. Secondly, I'll convert the number of occurrences of each pair of words to transition probabilities.

$$\text{Probability of each pair} = \frac{\text{Frequency of occurrence of start and end nodes together}}{\text{Total frequency of its Start Node}}$$

3. Build the Transition matrix to represent the transition probabilities of each pair of words.

$$P = \begin{bmatrix} p_{11} & p_{12} & . & . & . & p_{1r} \\ p_{21} & p_{22} & . & . & . & p_{2r} \\ . & . & & . & & . \\ . & . & & . & & . \\ p_{21} & p_{22} & . & . & . & p_{2r} \end{bmatrix}$$

4. After the completion of building the transition matrix, I'll initialize an input state vector of our choice to find the next possible outcome using:

$$\text{Next Sate} = \text{Initial State Vector} * \text{Transition Matrix}$$

5. Display the graph obtained using the transition matrix.

6. Generate the following outputs using the implemented model:
   - Next Possible Word.
   - Nth word from the starting word.
   - Generating a sentence with the starting word given as input.

The Markov chain is a perfect model for our text generator because our model will generate text using only the previous sequence of words. The advantage of using a Markov chain is that it's accurate, light on memory (only stores 1 previous state), and fast to execute.

4. Implementation

The implementation of Text Generator mainly involves three major steps:

1. Generating the lookup table.
2. Converting frequencies to transition probabilities.
3. Build the Markov Chains.

Generating the lookup table

First, I'll load the data from the text file and read each line of the text. Later I'll form a lookup table that stores the number of occurrences of each pair of words. Here the pair of words is nothing but the start node and the end node. So, let's name them as "start" and "end".

For Example, let's consider the first two sentences of our data containing, "what is your name", "what is your age".

Then the number of occurrences by pair of words would be:

what	is	2
is	your	2
your	name	1
your	age	1

Converting frequencies to transition probabilities

Once I have the start and end nodes and their frequencies, I'll find the probability of each pair occurring with respect to the starting node.

$$\text{Probability of each pair} = \frac{\text{Frequency of occurrence of start and end nodes together}}{\text{Total frequency of its Start Node}}$$

Let's consider the same example which I have used in the lookup table.

Frequency of occurrence of start and end nodes together is:

start	end	frequency
what	is	2
is	your	2
your	name	1
your	age	1

Frequency of each Start Node is:

start	frequency
what	2
is	2
your	2

Therefore, Probability of each pair is:

start	end	probability
what	is	1
is	your	1
your	name	0.5
your	age	0.5

Here's what the frequency and transition probability for first 20 pairs look like:

start	**end**	**frequency**	**transition probability**
are	they	1	0.11111111
are	you	6	0.66666667

are	your	2	0.22222222
can	I	2	1
do	dogs	1	0.08333333
do	fish	1	0.08333333
do	i	1	0.08333333
do	they	1	0.08333333
do	you	8	0.66666667
does	it	1	0.33333333
does	school	2	0.66666667
dogs	eat	1	1
favorite	color	1	0.33333333
favorite	food	1	0.33333333
favorite	song	1	0.33333333
fish	live	1	1
have	pets	1	0.5
have	sisters	1	0.5
how	are	2	0.33333333
how	is	1	0.16666667

Building the Markov Chains

As described in the above sections, I calculated the frequencies of each word in our data, its transition from one state to another, and the probabilities of each transition. Now, I need to build a transition matrix with all the required transition probabilities of the words calculated.

In the Markov process, I'll build the transition matrix to represent the transition probabilities of one state to another. It is denoted by P, and I calculate each value of the matrix using mathematical calculations. The transition matrix can be P can be written as:

$$P = \begin{bmatrix} p_{11} & p_{12} & . & . & . & p_{1r} \\ p_{21} & p_{22} & . & . & . & p_{2r} \\ . & . & & . & & . \\ . & . & & . & & . \\ p_{21} & p_{22} & . & . & . & p_{2r} \end{bmatrix}$$

For all the values in the matrix, pij≥0.

I can build a transition matrix using the probabilities I have, and I will get a transition matrix the same as the above. Using this transition matrix, I will build a directed weighted graph as shown in below fig 1.

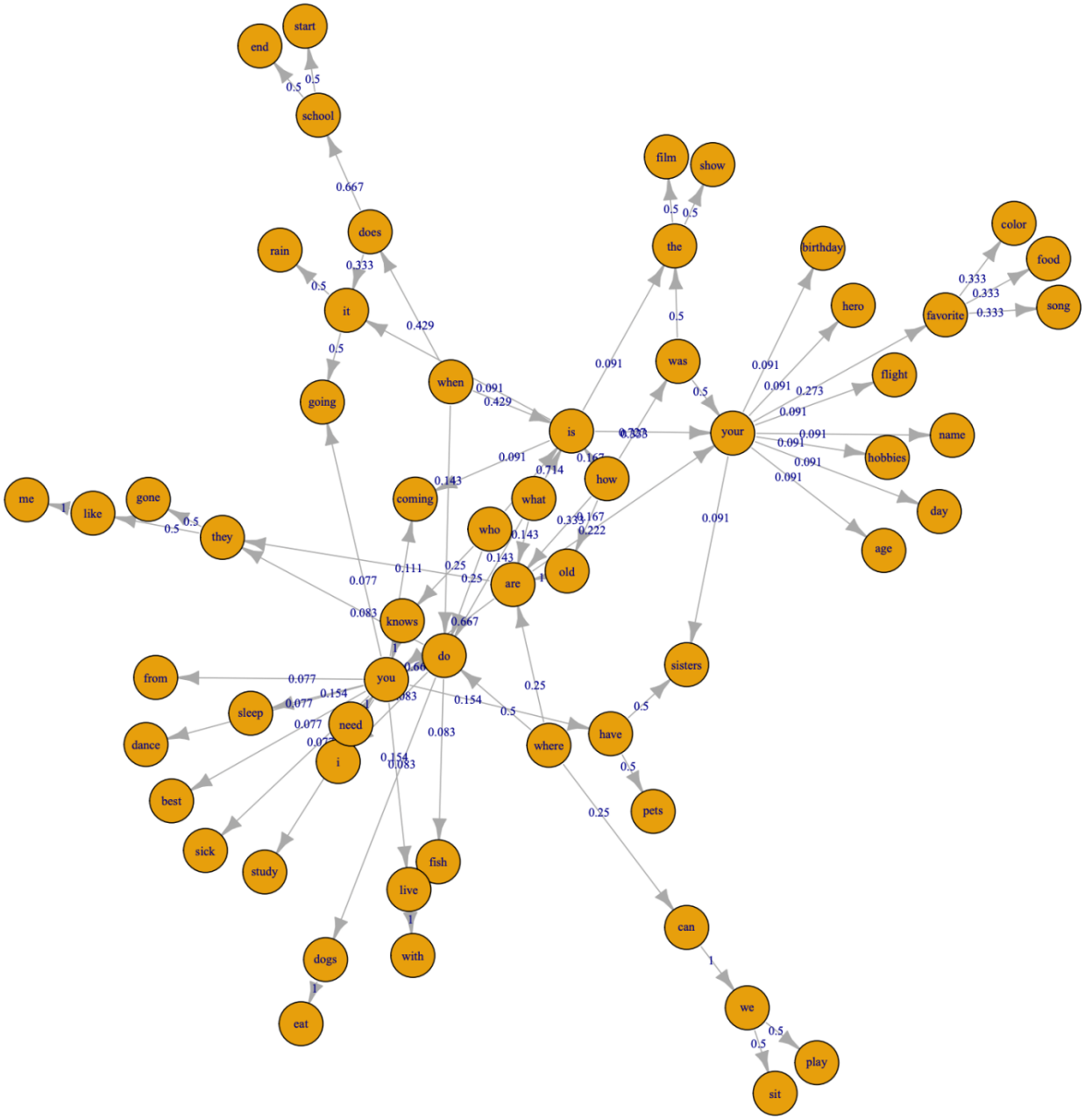

Fig 1. Directed weighted graph using transition probabilities.

After the completion of building the transition matrix, I'll initialize an input state vector to find the next possible outcome. The initial state vector contains the state of the input. In the initial state vector, I specify the current state of the input text by giving it a value of 1 and a value of 0 to all the other features in the vector.

Let's say I initialize a state vector with '10' features in it, and 'what' is one of the features in the vector. If I want to initialize the vector with the input 'what', then I will assign the value of 1 to 'what' and assign 0 to all the other features. In this way the initial state vector is initialized with the current state of the input.

As I have the initial state vector and the transition matrix, I will multiply both to predict the next possible state. This gives us the next possible text outcome provided with an input text.

Generating text

As mentioned above, I have a fully developed Markov chain model to generate/predict the different types of outcomes.

I am generating the next possible text/word based on the input given to the model, the Nth word from the starting point of the input, and I can also generate a full sentence based on the input.

The first type of outcome I generated using this model was to predict the immediate possible word given the input. Let's say I choose the text 'are' as my input and feed it to the model. Now, what the model does is initialize the state vector with the given input. Then it will multiply this vector with the transition matrix, and it takes all the probabilities of the next possible text into account. Now, from the probabilities I have at our disposal, the model will choose the text with the highest probability to predict the immediate possible outcome.

The second type of outcome I generated using this model was to predict the nth possible text given the input. In this case, I will initialize the model by providing the input to the state vector. Here I will multiply the state vector with the transition matrix n times. After multiple internal calculations, the model will have a set of probabilities. It will choose the text with the highest probability from the matrix as the nth possible outcome.

The third type of outcome I generated is to predict a full sentence based on the input text I provide to the model. In this case, I start with initializing the state vector with

the text input the same as before. After initializing the state vector, the model will multiply this state vector with a transition matrix, as I have already established. Now, based on this process, the model will predict the next outcome. Then the model will take the predicted text and will assign it to the state vector and repeat the process. This means that the outcome of the first input will be assigned back to the state vector, and it will act as an initial state vector in this step. Now, the model will predict the outcome of the next possible text using the newly assigned input text. This way, it will keep on iterating through the process till it creates a full sentence.

The examples and the outcomes generated can be found below in the appendix section.

5. Conclusion

Markov Chains is a simple but also effective model to implement text generator predictions. As it uses the probability of the transition between the words, the text generator can just be a clutter of words when creating a sentence with no context. But for effective text generation, the corpus needs to be filled with similar and contextual documents. Markov chains are the building blocks of other more sophisticated modeling techniques. Even though there are other techniques to implement text generator, Markov chains are much easier to implement.

References

https://towardsdatascience.com/structuring-text-with-graph-representations-41dd4f2a3ab3
https://towardsdatascience.com/text-generation-with-markov-chains-an-introduction-to-using-markovify-742e6680dc33
https://www.educative.io/blog/deep-learning-text-generation-markov-chains
Markov chain - Wikipedia
Transition probabilities - Encyclopedia of Mathematics

Appendix I (R Code)

```
# Install the required Packages
install.packages("hash")
install.packages("stringr")
install.packages("igraph")
install.packages("powerplus")
library(hash)
library(stringr)
library(igraph)
library(powerplus)

# Using hash to generate lookup table
hash_map=hash()
freq=hash()

# Initializing a dataframe having start node, end node and their edge weights
df=data.frame(matrix(ncol = 3, nrow = 0))
colnames(df)=c("start","end","edge_weight")

# Defining a function named Construct_weightedGraph
Construct_weightedGraph = function(data){
for (sent in data){
words = str_split(sent," ")[[1]]
for (i in 1:(length(words)-1)){
pair = paste(words[i],words[i+1])

# Calculating frequency of occurrence of start node
if(has.key(words[i],freq))
{
freq[words[i]] = values(freq,words[i])+1
}
else
{
freq[words[i]] = 1
}

# Calculating frequency of occurrence of start and end nodes together
if(has.key(pair,hash_map))
{
```

```
hash_map[pair] = values(hash_map,pair)+1
}
else
{
hash_map[pair] = 1
}
}
}
store_keys=keys(hash_map)
len = length(store_keys)
for(i in 1:len){
edge = str_split(store_keys[i]," ")[[1]]
weight = values(hash_map,store_keys[i])[[1]]/values(freq,edge[1])[[1]]
df[nrow(df)+1,]=c(edge[[1]],edge[[2]],as.numeric(weight))
}
return(df)
}

# Reading data from the text file
data = readLines(file.choose())
list=Construct_weightedGraph(data)
list[,"edge_weight"]= sapply(list[, "edge_weight"], as.numeric)

# Creating a graph from the dataframe
g=graph.data.frame(list[c("start","end")],directed=TRUE)
E(g)$weight=list$edge_weight
plot(g,vertex.size=125,vertex.label.cex=.7,edge.label.cex=.7,edge.label=round(E(g
)$weight,3),edge.arrow.size=.3,rescale = FALSE, xlim=c(-20,20))
adj_mat=as_adjacency_matrix(g,attr= "weight")
transition=as.matrix(adj_mat)
words=rownames(transition)

# Initializing state matrix with zeroes
state=matrix(0,ncol=dim(transition)[1])
state=state%*%transition

# Generating next occurrence using Markov Chains
print("Input the word:")
input = readline()
state[1,input]=1
```

```
trans_updated=state%*%transition
index = which(trans_updated==max(trans_updated))
sprintf("Next possible word is: %s",words[index])
sprintf("Probability of Occurrence is: %f", max(trans_updated))

# Generating possible 2nd word from the start word using Markov Chains
state=matrix(0,ncol=dim(transition)[1])
state=state%*%transition
print("Input the word:")
input = readline()
state[1,input]=1
trans_updated=state%*%Matpow(transition,2)
index = which(trans_updated==max(trans_updated))
sprintf("2nd possible word is: %s",words[index])
sprintf("Probability of Occurrence is: %f", max(trans_updated))

# Generating a sentence with given start word using Markov chains
state=matrix(0,ncol=dim(transition)[1])
state=state%*%transition
temp=state
print("Input the word:")
input = readline()
sent=input
state[1,input]=1
while(1){
trans_updated=state%*%transition
if(max(trans_updated)!=0){
index = which(trans_updated==max(trans_updated))
sent=paste(sent,words[index][1])
state=temp
state[1, words[index][1]]=1
}
else{
break
}
}
sprintf("Generated Sentence is: %s",sent)
```

Appendix II (Outputs)

Generating next occurrence using Markov Chains:

Example 1:

```
> input = readline()
are
> state[1,input]=1
> trans_updated=state%*%transition
> index = which(trans_updated==max(trans_updated))
> sprintf("Next possible word is: %s",words[index])
[1] "Next possible word is: you"
> sprintf("Probability of Occurrence is: %f", max(trans_updated))
[1] "Probability of Occurrence is: 0.666667"
```

Example 2:

```
> input = readline()
how
> state[1,input]=1
> trans_updated=state%*%transition
> index = which(trans_updated==max(trans_updated))
> sprintf("Next possible word is: %s",words[index])
[1] "Next possible word is: are" "Next possible word is: was"
> sprintf("Probability of Occurrence is: %f", max(trans_updated))
[1] "Probability of Occurrence is: 0.333333"
```

Generating possible 2nd word from the start word using Markov Chains

Example 1:

```
> input = readline()
when
> state[1,input]=1
> trans_updated=state%*%Matpow(transition,2)
> index = which(trans_updated==max(trans_updated))
> sprintf("2nd possible word is: %s",words[index])
[1] "2nd possible word is: your"
> sprintf("Probability of Occurrence is: %f", max(trans_updated))
[1] "Probability of Occurrence is: 0.311688"
```

Example 2:

```
> input = readline()
do
```

```
> state[1,input]=1
> trans_updated=state%*%Matpow(transition,2)
> index = which(trans_updated==max(trans_updated))
> sprintf("2nd possible word is: %s",words[index])
[1] "2nd possible word is: live"
> sprintf("Probability of Occurrence is: %f", max(trans_updated))
[1] "Probability of Occurrence is: 0.185897"
```

Generating a sentence with given start word using Markov Chains

Example 1:

```
> input = readline()
what
> sent=input
> state[1,input]=1
> while(1){
+ trans_updated=state%*%transition
+ if(max(trans_updated)!=0){
+ index = which(trans_updated==max(trans_updated))
+ sent=paste(sent,words[index][1])
+ state=temp
+ state[1, words[index][1]]=1
+ }
+ else{
+ break
+ }
+ }
> sprintf("Generated Sentence is: %s",sent)
[1] "Generated Sentence is: what is your favorite color"
```

Example 2:

```
> input = readline()
where
> sent=input
> state[1,input]=1
> while(1){
+ trans_updated=state%*%transition
+ if(max(trans_updated)!=0){
+ index = which(trans_updated==max(trans_updated))
+ sent=paste(sent,words[index][1])
```

```
+ state=temp
+ state[1, words[index][1]]=1
+ }
+ else{
+ break
+ }
+ }
> sprintf("Generated Sentence is: %s",sent)
[1] "Generated Sentence is: where do you have pets"
```