Project 1

Will Graham; Richelle Lee; Robin Lin

2022-10-06

# Project 1  
# Will Graham; Richelle Lee; Robin Lin  
# 2022-10-06  
### Group Contributions:  
# Instead of splitting tasks between our group, we worked collaboratively on each step of the process.  
# Contributions were 1/3 per team member.  
  
  
  
### Loads the Bible text and removes the unecessary pages.  
  
# User must set wd accordingly  
a <- scan("pg10.txt",what="character",skip=104) ## skip contents  
n <- length(a)  
a <- a[-((n-2886):n)] ## strip license  
a <- a[-grep("[0123456789]:[0123456789]",a)] ## strip out verse numbers  
### Giving punctuations their own indexes in bible text list.  
  
  
# Separating words and punctuation marks.  
indexes\_split\_punct <- function(string){  
 indexes <- grep("[',','.','?','!',':',';']",a) ## obtain the indexes of words containing specified punctuations characters   
 return(indexes)  
}  
  
  
# Returns the last character in a string.  
get\_punct <- function(r){  
 return(substr(r, nchar(r), nchar(r)))  
}  
  
  
# Splits punctuation marks from words, and inserts them into their correct positions in the final Bible text. Fills in remaining slots with words from the original Bible text.  
split\_punct <- function(b, test){  
 punct\_indexes <- indexes\_split\_punct(b)  
 punct <- lapply(b[punct\_indexes] , FUN = get\_punct)  
 new\_b <- rep(0, (length(punct\_indexes) + length(b)))  
 final\_punct\_indexes <- punct\_indexes + 1:length(punct\_indexes) ## Pushes punctuation indexes forward by n slots, where n is an accummulation of the number of punctuation marks.  
 new\_b[final\_punct\_indexes] <- punct  
 no\_punct\_list <- gsub('[[:punct:] ]+','',b)  
 if (test == TRUE){ ## Used for testing. Reduces sample size to 100.  
 b <- b[1:100]  
 new\_b <- new\_b[1:115]  
 }  
 new\_b[-final\_punct\_indexes] <- no\_punct\_list  
 return(new\_b)  
}  
### Lowering capitals and calculating the top 500 most common words.  
  
  
new\_b <- split\_punct(a, FALSE)  
lower\_new\_b <- tolower(new\_b)  
unique\_words <- unique(lower\_new\_b)  
index\_match <- match(lower\_new\_b, unique\_words) ## Mapping all Bible words to their unique entry in the unique list.  
tabulate\_index\_match <- tabulate(index\_match) ## Count the occurrences of the unique words in the Bible text.  
sort\_results <- sort(tabulate\_index\_match, index.return = TRUE, decreasing = TRUE)  
top\_500 <- sort\_results$ix[1 : 500] ## Pull the top 500 most common words.  
result\_list <- unique\_words[top\_500]  
### Saving: every occurrence of 3 consecutive common words, distinct combinations of two (and three) consecutive common words.  
### Counts occurrences of distinct combinations of two (and three) consecutive common words.  
  
  
# Saving every occurrence of 3 consecutive common words.  
common\_word\_match <- match(lower\_new\_b, result\_list)  
first\_column <- common\_word\_match[1 : (length(common\_word\_match) - 2)]  
second\_column <- first\_column[2 : (length(common\_word\_match) - 1)]  
third\_column <- first\_column[3 : length(common\_word\_match)]  
T\_array <- data.frame(first\_column, second\_column, third\_column)  
T\_array <- T\_array[is.na(rowSums(T\_array)) == FALSE,] ## Drop rows that contain uncommon words.  
  
  
# Maps indexes to common words  
Index\_to\_common\_words <- function(x){  
 return(result\_list[x])  
}  
  
  
# Calculating the probability of selecting any common words with no prior word.  
matrix\_x <- cbind(c(lapply(T\_array$first\_column, FUN = Index\_to\_common\_words)), c(lapply(T\_array$second\_column, FUN = Index\_to\_common\_words)), c(lapply(T\_array$third\_column, FUN = Index\_to\_common\_words))) ## Converts indexes to words  
Inital\_W\_map <- unique(T\_array$first\_column)  
Intial\_word\_probability <- tabulate(match(T\_array$first\_column, unique(T\_array$first\_column)))  
Initail\_prob <- Intial\_word\_probability/sum(Intial\_word\_probability) ## Probability weights for selecting common words with no prior words.  
  
  
# Counts occurrences of distinct combinations of two consecutive common words.  
second\_W\_map <- unique(T\_array[1:2]) ## All observed combinations of two consecutive common words.  
second\_W\_map\_conc <- paste(second\_W\_map[[1]], second\_W\_map[[2]]) ## Concatenated values for use in future mapping  
T\_array\_conc <- paste(T\_array[[1]], T\_array[[2]])  
df2 <- data.frame(c(second\_W\_map\_conc), rep(0, length(second\_W\_map\_conc[[1]]))) ## Initialises data frame to store distinct combination frequencies.  
colnames(df2) <- c("Concat\_w1\_w2", "Freq")  
match\_indexes <- match(T\_array\_conc, second\_W\_map\_conc)  
tabulate\_match\_index <- tabulate(match\_indexes) ## Count the occurrences of two consecutive common words in the Bible text via. T\_array.  
df2['Freq'] <- tabulate\_match\_index  
  
  
# Counts occurrences of distinct combinations of three consecutive common words.  
third\_W\_map <- unique(T\_array[1:3]) ## All observed combinations of three consecutive common words.  
third\_W\_map\_conc <- paste(third\_W\_map[[1]], third\_W\_map[[2]], third\_W\_map[[3]]) ## Concatenated values for use in future mapping  
T\_array\_conc2 <- paste(T\_array[[1]], T\_array[[2]], T\_array[[3]])  
df3 <- data.frame(c(third\_W\_map\_conc), rep(0, length(third\_W\_map\_conc[[1]]))) ## Initialises data frame to store distinct combination frequencies.  
colnames(df3) <- c("Concat\_w1\_w2\_w3", "Freq")  
match\_indexes2 <- match(T\_array\_conc2, third\_W\_map\_conc)  
tabulate\_match\_index2 <- tabulate(match\_indexes2) ## Count the occurrences of two consecutive common words in the Bible text via. T\_array.  
df3['Freq'] <- tabulate\_match\_index2  
### Defining functions to generate the next appropriate word in a Markov text model.   
### These functions handle cases when we have: no valid prior word, one valid prior word, and two valid prior word.  
  
  
# Generates a random common word with probability weights proportinal to the frequency of each common word in the Bible text.  
word\_1 <- function(punctuation){  
 PI <- grep("[',','.','?','!',':',';']", result\_list) ## Fetches indexes of specified punctuation marks from list of 500 most common word.  
 if (punctuation == TRUE){  
 return(as.numeric((sample(Inital\_W\_map, 1, prob = Initail\_prob))))  
 }  
 while (TRUE){ ## We do not want the first word to be a punctuation mark; pull another random word if a punctuation mark is selected.  
 ret <- sample(Inital\_W\_map, 1, prob = Initail\_prob)  
 if (!(ret %in% PI)) {  
 return(as.numeric(ret))  
 }  
 }  
}  
  
  
# Generates a random common word with probability weights proportinal to the frequency of possible word pairs (based on prior word) in the Bible text.   
word\_2 <- function(w1){  
 new\_ops <- second\_W\_map[second\_W\_map[1] == w1,] ## Possible unique common word pairs  
 freq <- c(1:length(new\_ops[[1]]))  
 for (i in c(1:length(new\_ops[[1]]))){  
 freq[i] <- subset(df2, df2$Concat\_w1\_w2 == paste(new\_ops[i,1], new\_ops[i,2]), 2)[[1]] ## Occurrences of common word pairs in the Bible text serve as probability weights.  
 }  
 sample\_1 <- as.numeric(sample(as.numeric(as.character(as.factor(new\_ops[,2]))), 1, prob = freq)) ## Return random word choice based on specified probabilities.  
 return(sample\_1)  
}  
  
  
# Generates a random common word with probability weights proportinal to the frequency of possible word triplets (based on prior word pairs) in the Bible text.   
word\_3 <- function(w1, w2){  
 new\_ops <- third\_W\_map[third\_W\_map[1] == w1 & third\_W\_map[2] == w2,] ## Possible unique common words triplets  
 freq <- c(1:length(new\_ops[[1]]))  
 for (i in c(1:length(new\_ops[[1]]))){  
 freq[i] <- subset(df3, df3$Concat\_w1\_w2\_w3 == paste(new\_ops[i,1], new\_ops[i,2], new\_ops[i,3]),2)[[1]] ## Occurrences of common word triplets in the Bible text serve as probability weights.  
 }  
 sample\_1 <- as.numeric(sample(as.numeric(as.character(as.factor(new\_ops[,3]))), 1, prob = freq)) ## Return random word choice based on specified probabilities.  
 return(sample\_1)  
}  
### Calculating proportion of capital occurrences for each common word in the Bible text.  
  
  
# Counting the occurrences of common word capitals in the Bible text.  
cap\_words <- grep("[A-Z]", new\_b, value = T)  
common\_cap\_words <- toupper(cap\_words[tolower(cap\_words) %in% result\_list]) ## If the word starts with a capital letter, capitalise the whole word and store it.  
unique\_cap\_words <- unique(common\_cap\_words)  
cap\_index\_match <- match(common\_cap\_words, unique\_cap\_words)  
freq\_cap\_index\_match <- tabulate(cap\_index\_match)  
cap\_freq\_array <- data.frame(Unique\_Cap\_Words = unique\_cap\_words, Frequency = freq\_cap\_index\_match) ## Store the number of times a common word appears with a capital letter in the Bible text.  
  
  
# Counting the occurences of common words (irrespective of capitals) in the Bible text.  
lower\_index\_match <- match(lower\_new\_b, result\_list) ## Maps indexes to every word in the Bible from the top 500 most common word list (where possible)  
freq\_lower\_index\_match <- tabulate(lower\_index\_match)  
lower\_freq\_array <- data.frame(Unique\_Lower\_Words = result\_list, Frequency = freq\_lower\_index\_match) ## Stores number of occurrences of top 500 most common words in the Bible text.  
cp <- rep(0, length(result\_list)) ## Initialise a list for storing capital common word proportions.   
for (i in 1:length(result\_list)){  
 if (!(toupper(result\_list[i]) %in% cap\_freq\_array$Unique\_Cap\_Words)){  
 cp[i] <- 0  
 }else{  
 ## Divide the number of times a common word appears as a capital in the Bible text by the number of times it appears, irrespective of capitalisation.  
 cp[i] <- cap\_freq\_array[cap\_freq\_array$Unique\_Cap\_Words == toupper(result\_list[i]),2]/lower\_freq\_array[lower\_freq\_array$Unique\_Lower\_Words == result\_list[i],2]  
 }  
}  
dfcp <- data.frame(result\_list, cp) ## Store the ratio of the capital occurrences of common words.  
### Generates a 50 word "paragraph" using a second order Markov model  
  
  
words\_generator\_50 <- function(){  
 sentence <- as.numeric(rep (0,50))  
 w1 <- word\_1(FALSE) ## Initialise the first word assuming no prior words.  
 sentence[1] <- w1  
 w2 <- word\_2(w1) ## Generate the second word based on the first word.  
 sentence[2] <- w2  
 ## Attempt to generate the next word in the sequence based on preceding word pair; if impossible, attempt to generate the next word in sequence based on preceding word; if impossible, generate the next word assuming no prior words.  
 for (i in c(3:length(sentence))){  
 if(paste(sentence[i - 2], sentence[i - 1]) %in% df2[1]){ ## Check if preceding word pair starts any common word triplets in the Bible text.   
 sentence[i] <- word\_3(sentence[i - 2], sentence[i - 1]) ## If yes, generate next word based on preceding word pair.  
 } else if (sentence[i - 1] %in% second\_W\_map[[1]]){ ## Check if preceding word starts any common word pairs in the Bible text.  
 sentence[i] <- word\_2(sentence[i - 1]) ## If yes, generate next word based on preceding word.  
 } else {  
 sentence[i] <- word\_1(FALSE) ## Default to random common word generation based on frequency in Bible text.  
 }  
 }  
   
   
 # Convert common word indexes to their respective words.  
 sentence2 <- c(1:50)  
 for (i in c(1:length(sentence2))){  
 if (dfcp[sentence[i],2] <= 0.5){ ## If the common word appears as a capital less than 50% of the time in the Bible text, store it as a lowercase.  
 sentence2[i] <- result\_list[as.numeric(sentence[i])]}  
 else { ## Else, capitalises the first letter of the common word  
 sentence2[i] <- paste(toupper(substr(result\_list[as.numeric(sentence[i])], 1,1)), substr(result\_list[as.numeric(sentence[i])], 2,nchar(result\_list[as.numeric(sentence[i])])), sep ="")  
 }  
 }  
 return(sentence2)  
}  
### Generates a 50 word "paragraph" using a 0th order Markov model  
  
  
words\_generator\_50\_random <- function(){  
 sentence <- as.numeric(rep (0,50))  
 for (i in 1:length(sentence)){ ## Instead of trying higher order prediction models, always assume no prior word.  
 sentence[i] <- word\_1(TRUE)  
 }  
   
   
 # Same process as previous common word conversion.  
 sentence2 <- c(1:50)  
 for (i in c(1:length(sentence2))){  
 if (dfcp[sentence[i],2] <= 0.5){  
 sentence2[i] <- result\_list[as.numeric(sentence[i])]}  
 else {  
 sentence2[i] <- paste(toupper(substr(result\_list[as.numeric(sentence[i])], 1,1)), substr(result\_list[as.numeric(sentence[i])], 2,nchar(result\_list[as.numeric(sentence[i])])), sep ="")  
 }  
 }  
 return(sentence2)  
}  
### Generate the 50 word "paragraph" with a 2nd order Markov model.  
test <- words\_generator\_50()  
cat(test)

## thy side , nor wisdom is come to bring you . who hath commanded to the people in the altar , and twelve men , and that were like unto him , and ye them : and the woman whom do evil spirit of these words which was in the

## daughter , stand in God ? do for the morning , or whether God : and they sat on the altar , and all the first day David arose , that the son of my chief priests which had no man among you . now , saying , and round  
### Generate the 50 word "paragraph" with a 0th order Markov model.  
test1 <- words\_generator\_50\_random()  
cat("\n", test1)

##   
## . pray and into hundred this to that head things of and his I because and against forth not but destroy are that my the us shall hand earth the know mighty whosoever in , , for shall I the . , , fruit ; month blessed his unto ,

##   
## , , . unto manner day fire rejoice the so be are the the and to therefore the you house do offer me and , , there : : gold , know blood and the me to ; under if land ? . of , send I evil let said