Scrape

Konnor Herbst

This is an explanation of the entire project and its usage, if you just want to learn how to use it, jump to the end

Prelim

To use this scraping program for websites we first have to go ahead and load the project into R so that we can run it. The program that actually does the scraping is called "news_scrape.R" and is listed below. You will want R and RStudio installed along with the rvest package in R, which you can install using the console command install.packages("rvest") directly from RStudio, also reading up on R data frames would be helpful, because that is how our output is formatted.

Build Explanation

The program is built systematically in a series of steps.

1) Find the urls on a specified site (usually a home page for a news site) and catalog them.
2) Continue to catalog urls from links found on other pages up to a certain depth, that just cataloging the home page urls is a depth of 1, but cataloging the urls found within the home page urls is a depth of 2.
3) Taking all unique urls the our pages link to and searching their source files for specific keywords or structural elements.

Step 1: Find the urls on a specified site.

Below is the function that allows us to enter the site and scrape its urls into a list. The function has arguments urls and chain_base. Urls is called with just the home page at the first step and depending on the source chain_base may be NA or a url chain specific to the site. The details don't really matter though, because I have cataloged all this data in a csv file in my github repo. In the end you will only need to call one function and specify row/col arguments from the csv file to scrape a specific source.

```
#'Finds all the urls that are associated with a url_root
#'(a base site) or a list of urls so that it can be called recursively
#'
#'@param urls(optional unless recursing) - a list of urls that we want to find links that chain directl
#'@param chain_base - whether the urls of a site are directly linked or not
#'
#'@return - all the urls associated with the url_root that base themselves off the url_root OR if urls
#'all the urls found within the pages of urls that base themselves off url_root
find_internal_urls <- function(urls, chain_base = NA) {
    # we first check if we are just doing the first call (from the base) or not
    final_url_links <- character()
    for (url in urls) {
        found_error <- FALSE
        # we catch any access errors in reading
        tryCatch({
            loaded_html <- read_html(url)
```

```
            possible_linked_urls <- loaded_html %>% html_nodes("a") %>% html_attr("href")
        }, warning = function(e) {
        }, error = function(e) {
            found_error <<- TRUE
        })
        if (found_error) {
            next
        }
        closeAllConnections()
        final_url_links <- c(final_url_links, possible_linked_urls[!is.na(possible_linked_urls)])
    }
    if (!is.na(chain_base)) {
        final_url_links <- (paste0(as.character(chain_base), final_url_links))
    }
    return(unique(final_url_links))
}
```

Step 2: Go to a specific depth for a site, that is if the home page is depth of 1, then the urls that are linked to from the homepage urls are depth of 2.

Below is the function that allows us to go a specific depth into our url cataloging, it works by uniquely cataloging links it hasn't found and returning all of them (all depths) into a single vector. This vector can then be used to scrape the individual pages/articles found at those urls for occurences of our keywords. In general a depth of 5 is not better than a depth of 10, but a depth of 10 takes over a day to run (b/c of the tree structure of the algorithm) and a depth of 5 only takes a few hours tops.

```
#'Searches the list of urls HTML for links to follow
#'
#'@param url_root - the base url for the sites that we want to include
#'example: 'https://example.com' would be the root for 'https://example.com/exampleChain'
#'@param depth - the amount of 'layers' that we want to parse i.e. the amount of times we want to
#'recursively call the function. We have this b/c after a few iteration we can have a massive amount of
#'and we might want to limit the search because of time constraints or possibly we aren't finding anyth
#'@param chain_base - whether the urls of a site are directly linked or not
#'
#'@return a vector of links that correspond to the links associated with a site
find_urls_single_source <- function(urls, depth, chain_base = NA) {
    # first we find the links from the home page
    complete_links <- character()
    level <- 1
    repeat {
        closeAllConnections()
        urls <- find_internal_urls(urls = urls, chain_base = chain_base)
        complete_links <- c(complete_links, unique(urls))
        level <- level + 1
        if (level > depth) {
            break
        }
    }
    return(unique(complete_links))
}
```

Step 3: Taking all unique urls the our pages link to and searching their source files for specific keywords or structural elements.

Below is really the meat of our analysis, it takes all the urls for a specific source and searches their source files for occurances of keywords(specified by the terms argument) and returns either NULL or some meta data/term counts if we find any of our terms in the url listing. Breaking down the arguments of the function you really only need to know about 3, url = the specific url the function is examining for our keywords (this is found by the pervious functions) source = The name of the source terms = our listing of terms that we want to search the url for

The other arguments are a little more complex, but I handle all of them and they are found in the meta data file english_identifiers.csv in my github repo title_selector = A specific way the source arranges the titles on its webpages published_selector = A specific way that the source arranges publishing date on its webpages text_selector = A specific way that the source arranges text (to be scraped) on its webpages

```
#'Searches the text of an article for occurences of our terms and return either NULL if there are none
#'the title, source, url, date of publishing and term counts if any of the terms are there
#'
#'@param url - the link to the site that we want to search
#'@param source - the name of the source (can be anything it is only for a direct return so we can easi
#'@param terms - a character vector of the terms we want to search for
#'@note - all selectors should be the least restrictive we can find i.e 'p' is preferred to 'p article'
#'@param title_selector - the CSS selector for the article title
#'@param published_selector - the CSS selector for the publishing date
#'@param text_selector - the CSS selector for the text of the article that we want to parse for our ter
#'
#'@return either NULL if none of our terms are in the article or the title, source, url,
#'date of publishing and term counts if any of the terms are there
article_analysis <- function(url, source, terms, title_selector, published_selector,
    text_selector) {
    if (substr(url, nchar(url) - 3, nchar(url)) == ".pdf") {
        print("pdf doc")
        return(NULL)
    }
    # (1) first we build regex for our terms so that they can be accurately
    # searched
    terms_regex <- character()
    for (term in terms) {
        terms_regex <- c(terms_regex, paste0("\\W*((?i)", term, "(?-i))\\W*"))
        # print('terms')
    }
    # (2) next we make an empty list to store our counts print('counts')
    term_counts <- vector("list", length(terms))
    names(term_counts) <- terms_regex
    # (3) now we are ready to actually read in the text of our article
    # print('loading')
    found_error <- FALSE
    tryCatch({
        loaded_html <- read_html(url)
        article_text <- loaded_html %>% html_nodes(text_selector) %>% html_text()
    }, warning = function(e) {
    }, error = function(e) {
        found_error <<- TRUE
    })
    if (found_error) {
```

```
    print("error in opening")
    return(NULL)
}
closeAllConnections()

# (4) now to check for our terms in the article I want to occurences and
# counts of singular words and of word combos first we do single words by
# splitting the text on one space and filtering the regex to single terms
# print('pairing')
paired_indices <- grep(" ", terms_regex)
# so at this point there are three possibilities and we handle them all
# possibility 1 - there are no paired words in our terms, so
# length(paired_indices) will be 0 possibility 2 - there are no single words
# in our terms, so length(paired_indices) will be length(terms_regex)
# possibility 3 - we have a mixture of single and paired words in our term
# set possiblity 1
if (length(paired_indices) == 0) {
    single_regex <- terms_regex
    single_words_in_article <- unlist(strsplit(article_text, " "))
    for (index in 1:length(single_regex)) {
        term_counts[[single_regex[index]]] <- length(grep(single_regex[index],
            single_words_in_article, value = FALSE))
    }
    # possibility 2
} else if (length(paired_indices) == length(terms_regex)) {
    paired_regex <- terms_regex[paired_indices]
    single_regex <- terms_regex[-(paired_indices)]
    # the words in the article
    single_words_in_article <- unlist(strsplit(article_text, " "))
    odd_words <- single_words_in_article[c(TRUE, FALSE)]
    even_words <- single_words_in_article[c(FALSE, TRUE)]
    # the words in pairs of two in the article
    odd_pairings <- paste(odd_words, even_words)
    even_pairings <- paste(even_words, odd_words)
    for (index in 1:length(paired_regex)) {
        term_counts[[paired_regex[index]]] <- length(grep(paired_regex[index],
            even_pairings, value = FALSE))
        term_counts[[paired_regex[index]]] <- term_counts[[paired_regex[index]]] +
            length(grep(paired_regex[index], odd_pairings, value = FALSE))
    }
    # possibility 3
} else {
    paired_regex <- terms_regex[paired_indices]
    single_regex <- terms_regex[-(paired_indices)]
    # the words in the article
    single_words_in_article <- unlist(strsplit(article_text, " "))
    odd_words <- single_words_in_article[c(TRUE, FALSE)]
    even_words <- single_words_in_article[c(FALSE, TRUE)]
    # the words in pairs of two in the article
    odd_pairings <- paste(odd_words, even_words)
    even_pairings <- paste(even_words, odd_words)
    for (index in 1:length(single_regex)) {
        term_counts[[single_regex[index]]] <- length(grep(single_regex[index],
```

```
                single_words_in_article, value = FALSE))
        }
        # now I check the word combos
        for (index in 1:length(paired_regex)) {
            term_counts[[paired_regex[index]]] <- length(grep(paired_regex[index],
                even_pairings, value = FALSE))
            term_counts[[paired_regex[index]]] <- term_counts[[paired_regex[index]]] +
                length(grep(paired_regex[index], odd_pairings, value = FALSE))
        }
    }
    title <- html_nodes(loaded_html, title_selector) %>% html_text()
    published_date <- html_nodes(loaded_html, published_selector) %>% html_text()
    # we return null if we did not find any of our terms
    terms_empty <- all(as.numeric(unlist(term_counts)) == 0)
    if (terms_empty) {
        return(NULL)
    }
    # we return the source, title, published date, url and term counts if any of
    # our terms are in the article
    names(term_counts) <- terms
    meta_data <- c(source, title, published_date, url)
    names(meta_data) <- c("source", "title", "publishing date", "url")
    print("found")
    return(c(meta_data, term_counts))
}
```

Usage

Now that we have got the details out of the way, the question is how do we use the program? Well I tried to make it as easy to use as possible and you really only need to do three things,

1) Install R and RStudio with the rvest package
2) Load in the data from the "english_identifiers.csv" file in the github repo
3) Run the source analysis function with arguments from the "english_identifiers.csv" file. The arguments url, source, title_selector, text_selector, chain_base are all found by indexing into the data frame created by reading in "english_identifiers.csv" while the terms of search (that is words we are searching articles for) and depth can be modified at will.

Example

Note: Our output is simply printed, but it is actually in a data frame object, and works just like an excel/csv file.

Below is the only function that you need to call after loading in the source_data using read.csv on "english_identifiers.csv" (this csv file contains everything we need except the terms and depth, and b/c it is a data frame we can index into it to call the function with specific args for a specific source.)

If we go for a depth of 1 with some example terms for our terms argument,

Here is the function and an example of loading "english_identifiers.csv"

```
source_analysis <- function(url, source, terms, title_selector, published_selector,
    text_selector, chain_base = NA, depth) {
    discovered <- find_urls_single_source(urls = url, depth = depth, chain_base = chain_base)
    source_hits <- character()
    for (link in discovered) {
        hit <- article_analysis(url = link, source = source, terms = terms,
            title_selector = title_selector, published_selector = published_selector,
            text_selector = text_selector)
        if (is.null(hit)) {
            next
        }
        print("found one")
        source_hits <- c(source_hits, hit)
    }
    return(source_hits)
}


source_data <- read.csv("/Users/konnorherbst/williams/godlonton_scrape/english_identifiers.csv",
    header = TRUE)
```

If we go for a depth of 1 with some example terms.

```
hits <- source_analysis(url = source_data[[2]][1], source = source_data[[1]][1],
    terms = c("prince, saudi", "federal"), title_selector = source_data[[4]][1],
    published_selector = source_data[[5]][1], text_selector = source_data[[6]][1],
    chain_base = source_data[[3]][1], depth = 1)

# hits # the entire data frame is printed sequentially

# Source Title 1 saudi_news_agency European shares open lower 2
# saudi_news_agency GOP, White House eye deep cuts to corporate tax rate 3
# saudi_news_agency Crown Prince congratulates German Chancellor on her
# victory in parliamentary elections 4 saudi_news_agency Custodian of the
# Two Holy Mosques congratulates German Chancellor on her victory in
# parliamentary elections 5 saudi_news_agency European shares open lower 6
# saudi_news_agency GOP, White House eye deep cuts to corporate tax rate 7
# saudi_news_agency Crown Prince congratulates German Chancellor on her
# victory in parliamentary elections 8 saudi_news_agency Custodian of the
# Two Holy Mosques congratulates German Chancellor on her victory in
# parliamentary elections Date 1 \n Tuesday 1439/1/6 - 2017/09/26 2 \n
# Tuesday 1439/1/6 - 2017/09/26 3 \n Monday 1439/1/5 - 2017/09/25 4 \n
# Monday 1439/1/5 - 2017/09/25 5 \n Tuesday 1439/1/6 - 2017/09/26 6 \n
# Tuesday 1439/1/6 - 2017/09/26 7 \n Monday 1439/1/5 - 2017/09/25 8 \n
# Monday 1439/1/5 - 2017/09/25 URL prince, saudi federal 1
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1671076 0 1 2
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670990 0 2 3
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670959 0 1 4
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670958 0 1 5
# http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1671076#1671076 0 1
# 6 http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670990#1670990 0
# 2 7 http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670959#1670959
# 0 1 8
# http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670958#1670958 0 1
```

If we go for a depth of 5

```
hits <- source_analysis(url = source_data[[2]][1], source = source_data[[1]][1],
    terms = c("prince, saudi", "federal"), title_selector = source_data[[4]][1],
    published_selector = source_data[[5]][1], text_selector = source_data[[6]][1],
    chain_base = source_data[[3]][1], depth = 5)

hits

# Source Title 1 saudi_news_agency European shares open lower 2
# saudi_news_agency GOP, White House eye deep cuts to corporate tax rate 3
# saudi_news_agency Crown Prince congratulates German Chancellor on her
# victory in parliamentary elections 4 saudi_news_agency Custodian of the
# Two Holy Mosques congratulates German Chancellor on her victory in
# parliamentary elections 5 saudi_news_agency European shares open lower 6
# saudi_news_agency GOP, White House eye deep cuts to corporate tax rate 7
# saudi_news_agency Crown Prince congratulates German Chancellor on her
# victory in parliamentary elections 8 saudi_news_agency Custodian of the
# Two Holy Mosques congratulates German Chancellor on her victory in
# parliamentary elections 9 saudi_news_agency General / Wakil Putra Mahkota
# dan Presiden Somalia membahas hubungan bilateral dan perkembangan kawasan
# Date 1 \n Tuesday 1439/1/6 - 2017/09/26 2 \n Tuesday 1439/1/6 -
# 2017/09/26 3 \n Monday 1439/1/5 - 2017/09/25 4 \n Monday 1439/1/5 -
# 2017/09/25 5 \n Tuesday 1439/1/6 - 2017/09/26 6 \n Tuesday 1439/1/6 -
# 2017/09/26 7 \n Monday 1439/1/5 - 2017/09/25 8 \n Monday 1439/1/5 -
# 2017/09/25 9 \n enin 1438/6/7 - 2017/03/06 URL prince, saudi federal 1
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1671076 0 1 2
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670990 0 2 3
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670959 0 1 4
# http://www.spa.gov.sa//viewfullstory.php?lang=en&newsid=1670958 0 1 5
# http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1671076#1671076 0 1
# 6 http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670990#1670990 0
# 2 7 http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670959#1670959
# 0 1 8
# http://www.spa.gov.sa/viewfullstory.php?lang=en&newsid=1670958#1670958 0 1
# 9 http://www.spa.gov.sa//viewfullstory.php?lang=id&newsid=1598923 0 1
```

As can be seen, the output for a depth of 5 is just one more obervsation for a depth of 1. However, it takes much longer (~200x) than a depth of one. As a result of this I recommend running it with a depth of up to 5, but more than 5 would be prohibiteively slow with on new information. 4.)