BUILDING A MINI SEARCH ENGINE

Web Crawling and Preprocessing

By

Rajendra Banjade

The University of Memphis, TN, USA

2012

Automatically collects from <given website> 1500 unique documents. The documents should be proper (>50 valid tokens) .html, .txt, and .pdf files. Only retain from these documents the title, time date, and body. That is, the output will be a set of 1500 text files and not html docs.

```
# Automatically collects from memphis.edu 1500 unique documents.
# The documents (.html, .txt, .pdf) should  have >50 valid tokens
# .html, .txt, and .pdf files. Only retains from these documents the
# title, time&date, and body.
#
# VERSION HISTORY:
# Rajendra Banjade 11/04/2012
# The University of Memphis, TN, USA
#

require "stemmer.pl";          # porter stemmer
(http://tartarus.org/~martin/PorterStemmer/)
use LWP::Simple;               # utility package (use get() to fetch content
of url etc).
use CAM::PDF;                  # to process pdf files.
use CAM::PDF::PageText;        # for pdf to text conversion.

$baseUrl = "http://<a website url>";  # home page

my $WAITTIME = 2;             # wait time in second, be web server friendly.
My MAXDOCUMENT = 1500;
my $crawledPageCount = 0;     # counter for pages crawled (not all links be
crawled - duplication, anchor links etc are skipped).
my %uniqueLinkTable = ();     # unique links crawled.
my $link = "";
my $validDocCount = 0;        # valid documents (>50 valid tokens), .pdf,
.txt, or .html
my @links;
my @linkQueue = qw( lpr mcopy ps );  # queue of links, implementing Breadth
first crawling.
# for stopword removal
%stopWords = {};

# initialize the porter stemmer
&initialise();

# module to trim any whitespace in the string.
sub trim($)
{
    my $s = shift;
    $s =~ s/^\s+//;
    $s =~ s/\s+$//;
    return $s;
}

# Populate stop word hash with the list of words from a file.
open(SWFILE, "<stopwords.txt" )  or die("Failed to read stop word file:
stopwords.txt\n");
while (<SWFILE>) {
```

```
    $stopWords{&trim($_)} = 1;
}
close (SWFILE);

# Parses the page content, and preserve the valid words only.
sub processData{
    my ($pageData, $url, $localPath) = @_;

    # try to get the title and the date time, if exists. Can't get that from
PDF and text documents
    my $pageTitle = "NA";
    my $pageLastModified = "NA";
    my $pageBody = "";

    if ($url =~ /[\.txt|\.pdf]+$/){ # if text or pdf, don't need to check for
title, body etc.
        $pageBody = $pageData;
    } else {                          # if html file
        my @temp =();
        @temp = $pageData =~ /<title>(.*?)<\/title>/gi; # get the title of
the page (if exists).
        $pageTitle = shift @temp;
        @temp = $pageData =~ /<meta http-equiv="Last-Modified"
content="(.*?)">/gi; # get the last modified date time stamp
        $pageLastModified  = shift @temp;
        @temp = $pageData =~ /<body>(.*?)<\/body>/i;  # get the body text
        $pageBody = shift @temp;
        $pageBody =~ s/<!--.+?-->/ /g;
        $pageBody =~ s/<script.+?<\/script>/ /g;
        $pageBody =~ s/<.+?>/ /g;                # remove valid html tags
(opening | closing), do shortest match (?).
        $pageBody =~ s/ //g;               # remove nbsp;
        $pageBody =~ s/(http:\/\/|ftp:\/\/)?(\.?\w+-
*\w+)+\.(com|net|org|gov|edu)\/*//gi; # remove urls in text, <taken from
web>.
        $pageBody =~ s/mailto://g;              # remove mailto:
        #$fileContent =~ s/(.+)@(.+)\.(.{2,4})?//g;     # simple regex to
remove some email addresses, maynot match all email addresses.
    }
    $pageBody =~ s/[0-9]+//g;               # remove any digits
    $pageBody =~ tr/[A-Z]/[a-z]/;          # change everything to lowercase
    $pageBody =~ s/[[:punct:]]//g;          # remove puncutations
    $pageBody =~ s/[||||]?//g;           # remove quotes, they can still there.
    $pageBody =~ s/\s+/ /g;                 # replace any sequence of
whitespaces by a single whitespace (for easy split)

    my @tokens = split /\s+/, $pageBody;    # split by whitespace(s)

    # remove stop words as well
    my @validTokens = ();
    # foreach token, do not add to index words if it is an stopword.
    foreach $token (@tokens) {              # Case sensitive, match.
        if (! (exists $stopWords{$token})){
            $token = &stem ($token);
            if($token){
                push @validTokens, $token;
            }
```

```
        }
    }

    my $tokenCount = @validTokens;            # these are the valid tokens
(before removing stop words)

    # Do not count the document if it is too short (having 50 or less valid
tokens)
    if ($tokenCount <= 50) {
        return;
    }
    # create a file and save link, page title, and last modified time stamp.
    next unless (open (FILE, ">processed/$crawledPageCount.txt")); # if can't
save, skip.
    print FILE $pageBody;
    print FILE "\n\n\n====================";
    print FILE "<doc>\n<title>$pageTitle</title>\n<last-
modified>$pageLastModified</last-modified>";
    print FILE "\n<path>\n<local>$localPath</local>\n<web>$url</web></path>";
    print FILE "\n<body>".join(" ", @validTokens )."\n</body></doc>";
    close (FILE);

    $validDocCount++;
    print "\n $validDocCount: Link(valid page) processed: $url";

}

# starts from here
unshift (@linkQueue, $baseUrl);         # start from home page.
$uniqueLinkTable{$baseUrl} = 1;         # mark the home page as queued, avoids
redundancy.

while ($validDocCount < $MAXDOCUMENT) {
    $currentUrl = pop @linkQueue;       # get the link from the queue
    my $pageContent = "";
    @links = ();

    # TODO: if the page is robot.txt, avoid processing forbidden links.

    $crawledPageCount++;      # prevent id clash. So increment first and
subsequent skip/continue doesn't affect.
    my $fileName = "";
    # Handle pdf file, differently.
    if ($currentUrl =~ /\.pdf$/){
        $fileName = "raw/$crawledPageCount.pdf";      # give id to the file.
        my $tempTextFile = "temp.txt";                # temporary file for pdf
to .txt
        $pageContent = get ($currentUrl) || next;     # fetch the pdf content,
otherwise skip.
        next unless (open (RAWFILE, ">$fileName"));  # if can't create file,
skip.
        binmode(RAWFILE);                             # pdf must be saved as
binary file !!!
        print RAWFILE $pageContent;
        close (RAWFILE);
        my $pdf = CAM::PDF->new($fileName);           # read pdf file.
        my $pageCount = $pdf->numPages();                # get number of
```

```
pages
        next unless (open (TEMPFILE, ">$tempTextFile")); # if can't create
temp file, just skip that pdf.
        # Convert each pdf page to txt and Append to.
        foreach my $pageNum (1..$pageCount) {
            my $page = $pdf->getPageContentTree($pageNum);
            print TEMPFILE CAM::PDF::PageText->render($page);
        }
        close (TEMPFILE);
        next unless (open (TEMPFILE, "<$tempTextFile")); # if can't read
converted text, just skip
        my @content = <TEMPFILE>;
        $pageContent = join " ",@content;    # change to single string (space
separated for each line).
                                             # Just to process like other HTML
page's content.
        close (TEMPFILE);
        #remove temp .txt file.
        unlink($tempTextFile);
    }
    else {
        $pageContent = get ($currentUrl) || next;        # fetch the pdf
content, otherwise skip.
        #save file in a folder (with proper extension)
        if ($currentUrl =~ /[\.txt]+$/){
            $fileName = "raw/$crawledPageCount.txt";
        } else {
            $fileName = "raw/$crawledPageCount.html";
        }

        next unless (open (RAWFILE, ">$fileName"));    # if can't create
file, skip.
        no warnings;
        print RAWFILE $pageContent;
        close (RAWFILE);
        # make single line for preprocessing.. regular expressions fail if
spreaded over multiple line.
        $pageContent =~ tr{\r\n}{ };

        @links = $pageContent =~ /href="(.*?)"/gi; # get all (g) the url's
the page linked to the current page.
    }

    # for each link, either discard or add in the queue.
    foreach $link (@links) {
        if (!($link =~ /(\.pdf|\.html|\.htm|\.txt)+$/)) {  # if not linked to
pdf, txt, or html, just skip.
            next;
        }

        if ($link =~ /^\//) {       # if relative path, change to absolute.
            $link = $currentUrl.$link;
        }
        next unless ($link =~ /memphis\.edu/);           # skip, if linked to
external site.

        next unless (! (exists $uniqueLinkTable{$link}));   # Avoid redundant
```

```
processing
        unshift (@linkQueue, $link);      # add this link in the queue.
    }
    # parse and process the page content.
    &processData ($pageContent, $currentUrl,$fileName);
    # wait for some time before making another request to the server.
    sleep ($WAITTIME);  # be web server friendly.
}
```

Memphis.edu.crawler.pl

NOTES:

- ⇨ To process the pdf file, external modules CAM::PDF and CAM::PDF::PageText (both from CPAN) has been used in this program. They are useful while converting the pdf file to plain text and then preprocess.
- ⇨ Change the time interval of two consecutive requests to the server by changing the global variable $WAITTIME (set to 2 second).
- ⇨ Change the number of valid documents to download by changing the value of variable $MAXDOCUMENT (now set to 1500).

To install CAM::PDF module,

- ⇨ Go to CPAN client (in windows 7, please run it as administrator).
- ⇨ Issue a command, install CAM::PDF

To run the program,

- ⇨ Go to the <your folder>
- ⇨ Issue command $ perl Crawler.pl > output.txt
  Since the output is very large, it would be a good idea to redirect to some file (as shown above) and see.

The program,

a. Adds the home page in the queue of links to be retrieved.
b. Get the link from queue. Gets all the links directly linked from the page and adds to in the queue (i.e. do breadth first crawling). Processes the page and creates a text file if the page contains >50 words. Also saves the link to the site, local page, title, and last modified date (if any).
c. Repeat the process (b) until the valid document count is less than 1500.

Sample output:

The preprocessed file content would look like,

```
<doc>
<title>The University of Memphis :: Current Students :: University of
Memphis</title>
<last-modified>2012-05-18T20:42:00Z</last-modified>
<path>
<local>raw/6.html</local>
<web>http://memphis.edu/students.htm</web></path>
<body>skip navig skip main content skip search form search search type site
peopl futur student current student alumni friend faculti staff parent
visitor academ admiss research librari athlet give resourc media room az list
quick link email bursar offic event calendar job ecoursewar um elearn rodp
bookstor tom employe servic campu map librari fedex institut transcript
umdriv iam polic servic contact current student featur link academ academ
advis academ calendar academ polici academ stand good
standingwarningprobationsuspens catalog undergradu catalog graduat catalog
law school curriculum commenc graduat ceremoni diploma info degre audit track
progress degre educ tutor academ gener student popul final exam schedul gener
educ progra
</body></doc>
```