MAKE CS466 GREAT AGAIN

Lianqiang Mao

Vishwesh Majithia

prepared for

Dr.Yarowsky

May 14, 2016

For the final project we decided to analyze and compare the speech of politicians. We built several bots that scraped tweets, speech transcripts and also quotes by these politicians that happened to be mentioned in New York Times articles. Each of these bots had its unique list of implementation challenges

We choose to analyze three politicians -- Donald Trump, Hillary Clinton and Obama.

1 OVERVIEW

1.1 NEW YORK TIMES BOT

We extended our bot from hw4. We would used words such as ‘stated’, ‘said’, ‘claimed’ etc near the names of the politicians we add the text inside the quotation marks to our log. We looped over the document three times with different regex options.

For example:

```
$content=~s/Trump\s(said|added|claimed|stated|asked|exclaimed)[\s\w]*(\:|\,).\“([\-\’\.\,\?\!\s\w]+)\”//
```

This takes care of the scenario -Trump said”TEXT”

```
$content=~s/\“([\-\’\.\,\?\!\s\w]+)\”[\s\w\,]*(said|added|claimed|stated|asked|exclaimed)[\s\w(Mr\.)]*Trump//
```

This takes care of the scenario- “TEXT” said Trump

```
$content =~ s/\“([\-\’\.\,\?\!\s\w]+)\”[\s\w(Mr\.)]*Trump[\s\w]*(said|added|claimed|stated|asked|exclaimed)//
```

This takes care of the scenario- “TEXT” Trump said.

We ran into some issues however. And had difficulty figuring out how to extract the quotes in the following situations

->“For me, it worked very well,” Mr. Trump said. “For Fred, it wasn’t something that was going to work.”

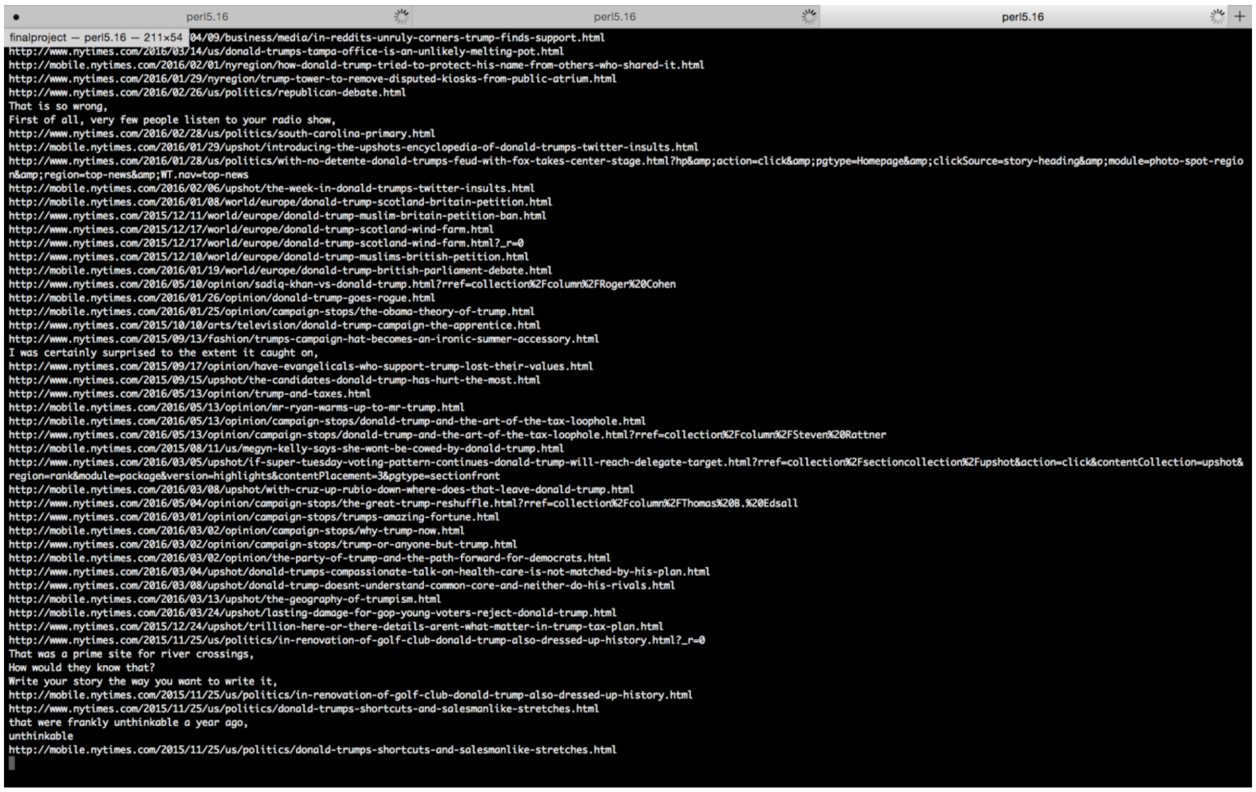
In this situation there was no way for our algorithm to find out who said the second sentence.

->“My net worth fluctuates, and it goes up and down with markets and with attitudes and with feelings — even my own feelings — but I try,” he once said.

Places where there were just pronouns were hard to implement because of the uncertainty that some else in the article might have said it and not the Politicians.

->Another issue we ran into was spouses. Here the statements made were about Donald Trump but they were made by his wife, Malania.

We made three different bots with the same basic structure called robot_hillary.pl robot_obama.pl and robot_trump.pl which are running simultaneously.



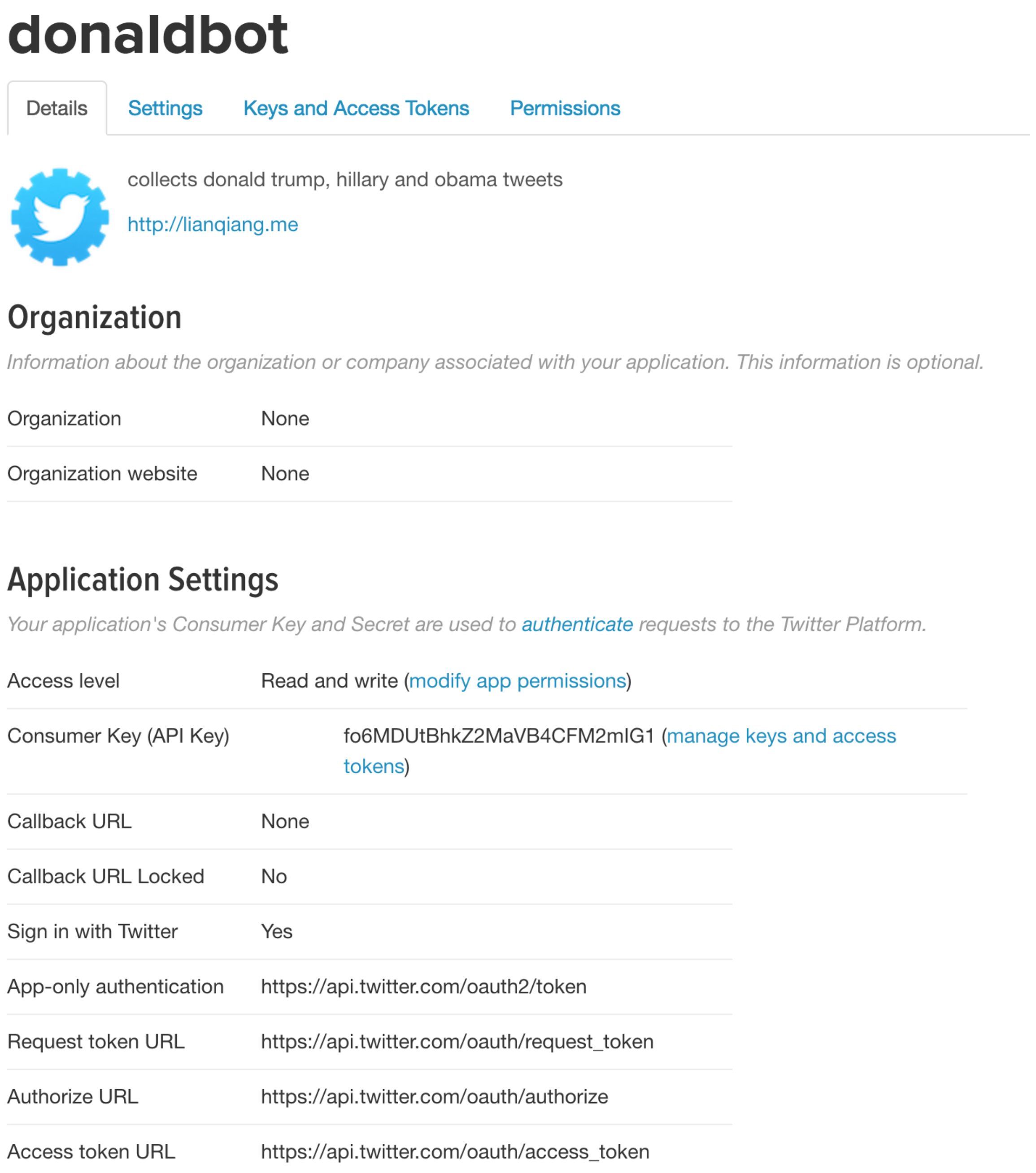
```
perl5.16                    perl5.16                    perl5.16
finalproject — perl5.16 — 211×54 04/09/business/media/in-reddits-unruly-corners-trump-finds-support.html
http://www.nytimes.com/2016/03/14/us/donald-trumps-tampa-office-is-an-unlikely-melting-pot.html
http://mobile.nytimes.com/2016/02/01/nyregion/how-donald-trump-tried-to-protect-his-name-from-others-who-shared-it.html
http://www.nytimes.com/2016/01/29/nyregion/trump-tower-to-remove-disputed-kiosks-from-public-atrium.html
http://www.nytimes.com/2016/02/26/us/politics/republican-debate.html
That is so wrong,
First of all, very few people listen to your radio show,
http://www.nytimes.com/2016/02/28/us/politics/south-carolina-primary.html
http://mobile.nytimes.com/2016/01/29/upshot/introducing-the-upshots-encyclopedia-of-donald-trumps-twitter-insults.html
http://www.nytimes.com/2016/01/28/us/politics/with-no-detente-donald-trumps-feud-with-fox-takes-center-stage.html?hp&amp;action=click&amp;pgtype=Homepage&amp;clickSource=story-heading&amp;module=photo-spot-regio
n&amp;region=top-news&amp;WT.nav=top-news
http://mobile.nytimes.com/2016/02/06/upshot/the-week-in-donald-trumps-twitter-insults.html
http://mobile.nytimes.com/2016/01/08/world/europe/donald-trump-scotland-britain-petition.html
http://www.nytimes.com/2015/12/11/world/europe/donald-trump-muslim-britain-petition-ban.html
http://www.nytimes.com/2015/12/17/world/europe/donald-trump-scotland-wind-farm.html
http://www.nytimes.com/2015/12/17/world/europe/donald-trump-scotland-wind-farm.html?_r=0
http://www.nytimes.com/2015/12/10/world/europe/donald-trump-muslims-british-petition.html
http://mobile.nytimes.com/2016/01/19/world/europe/donald-trump-british-parliament-debate.html
http://www.nytimes.com/2016/05/10/opinion/sadiq-khan-vs-donald-trump.html?rref=collection%2Fcolumn%2FRoger%20Cohen
http://mobile.nytimes.com/2016/01/26/opinion/donald-trump-goes-rogue.html
http://mobile.nytimes.com/2016/01/25/opinion/campaign-stops/the-obama-theory-of-trump.html
http://www.nytimes.com/2015/10/10/arts/television/donald-trump-campaign-the-apprentice.html
http://www.nytimes.com/2015/09/13/fashion/trumps-campaign-hat-becomes-an-ironic-summer-accessory.html
I was certainly surprised to the extent it caught on,
http://www.nytimes.com/2015/09/17/opinion/have-evangelicals-who-support-trump-lost-their-values.html
http://www.nytimes.com/2015/09/15/upshot/the-candidates-donald-trump-has-hurt-the-most.html
http://www.nytimes.com/2016/05/13/opinion/trump-and-taxes.html
http://mobile.nytimes.com/2016/05/13/opinion/mr-ryan-warms-up-to-mr-trump.html
http://mobile.nytimes.com/2016/05/13/opinion/campaign-stops/donald-trump-and-the-art-of-the-tax-loophole.html
http://www.nytimes.com/2016/05/13/opinion/campaign-stops/donald-trump-and-the-art-of-the-tax-loophole.html?rref=collection%2Fcolumn%2FSteven%20Rattner
http://mobile.nytimes.com/2015/08/11/us/megyn-kelly-says-she-wont-be-cowed-by-donald-trump.html
http://www.nytimes.com/2016/03/05/upshot/if-super-tuesday-voting-pattern-continues-donald-trump-will-reach-delegate-target.html?rref=collection%2Fsectioncollection%2Fupshot&action=click&contentCollection=upshot&
region=rank&module=package&version=highlights&contentPlacement=3&pgtype=sectionfront
http://mobile.nytimes.com/2016/03/08/upshot/with-cruz-up-rubio-down-where-does-that-leave-donald-trump.html
http://www.nytimes.com/2016/05/04/opinion/campaign-stops/the-great-trump-reshuffle.html?rref=collection%2Fcolumn%2FThomas%20B.%20Edsall
http://www.nytimes.com/2016/03/01/opinion/campaign-stops/trumps-amazing-fortune.html
http://mobile.nytimes.com/2016/03/02/opinion/campaign-stops/why-trump-now.html
http://www.nytimes.com/2016/03/02/opinion/campaign-stops/trump-or-anyone-but-trump.html
http://mobile.nytimes.com/2016/03/02/opinion/the-party-of-trump-and-the-path-forward-for-democrats.html
http://www.nytimes.com/2016/03/04/upshot/donald-trumps-compassionate-talk-on-health-care-is-not-matched-by-his-plan.html
http://www.nytimes.com/2016/03/08/upshot/donald-trump-doesnt-understand-common-core-and-neither-do-his-rivals.html
http://mobile.nytimes.com/2016/03/13/upshot/the-geography-of-trumpism.html
http://mobile.nytimes.com/2016/03/24/upshot/lasting-damage-for-gop-young-voters-reject-donald-trump.html
http://www.nytimes.com/2015/12/24/upshot/trillion-here-or-there-details-arent-what-matter-in-trump-tax-plan.html
http://www.nytimes.com/2015/11/25/us/politics/in-renovation-of-golf-club-donald-trump-also-dressed-up-history.html?_r=0
That was a prime site for river crossings,
How would they know that?
Write your story the way you want to write it,
http://mobile.nytimes.com/2015/11/25/us/politics/in-renovation-of-golf-club-donald-trump-also-dressed-up-history.html
http://www.nytimes.com/2015/11/25/us/politics/donald-trumps-shortcuts-and-salesmanlike-stretches.html
that were frankly unthinkable a year ago,
unthinkable
http://mobile.nytimes.com/2015/11/25/us/politics/donald-trumps-shortcuts-and-salesmanlike-stretches.html
```

1.2 TWITTER BOT

For this part of the project we tried using an LWP bot but twitter recently changed the way its bots work and ask you to register and authenticate your bots. There are private keys as well as authorization tokens which help enhance security. We ended up using the twitter API through a perl module called Net::Twitter::Lite::WithAPIv1_1

The bot we made is called donaldbot.

This allows us to search for queries in the twitter database. You can add parameters in the query such as “to:SELECTED_USER”, “from:SELECTED_USER”, “@SELECTED_USER” “lang=>en ” etc



donaldbot

collects donald trump, hillary and obama tweets

http://lianqiang.me

Organization

Information about the organization or company associated with your application. This information is optional.

Organization	None
Organization website	None

Application Settings

Your application's Consumer Key and Secret are used to authenticate requests to the Twitter Platform.

Access level	Read and write (modify app permissions)
Consumer Key (API Key)	fo6MDUtBhkZ2MaVB4CFM2mIG1 (manage keys and access tokens)
Callback URL	None
Callback URL Locked	No
Sign in with Twitter	Yes
App-only authentication	https://api.twitter.com/oauth2/token
Request token URL	https://api.twitter.com/oauth/request_token
Authorize URL	https://api.twitter.com/oauth/authorize
Access token URL	https://api.twitter.com/oauth/access_token

Our bot was registered and given permission so that we would be good robot citizens.

1.3 Speech Transcript

We got two speeches for each politician from the Internet. The link of the speeches are as follows:

Trump1: http://tinyurl.com/hwh3ulb

Trump2: http://tinyurl.com/p2clc8e

Obama1: http://tinyurl.com/yfsbhml

Obama2: http://tinyurl.com/kh92866

Hillary1: http://tinyurl.com/z8rtmyc

Hillary2: http://tinyurl.com/jcm2oka

For each transcript, we removed terms like "(APPLAUSE)", "TRUMP: ", etc. As the speeches made by each politician are sufficiently long and do not have words from other people, they are good benchmarks to be compared with the result of tweets and quotes from New York Times.

2 Result Analysis

We used many different metrics to measure their speeches for complexity. The link to raw data can be found at: https://docs.google.com/spreadsheets/d/1VoUXiSD4q5GPMMAixfitx0cgLE-oz3wjEM646xG6CcQ/edit?usp=sharing.

2.1 Total words

For speeches, the total words are all over 1000, with Trump's two speeches more than 2500. For tweets, Obama has the smallest number of 229 words. And for New York Times, it is very hard to get posts about Obama and retrieve his words from them. Thus we only have 225 words from Obama, with more than 1000 from Trump and Hillary each.

2.2 Total sentences

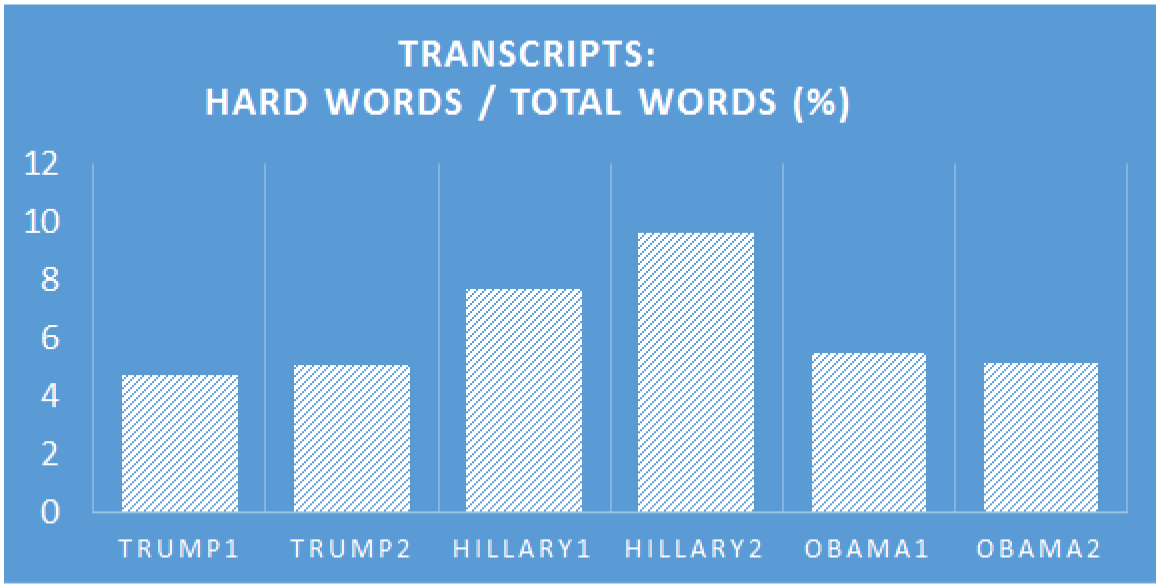
We count sentences with the number of ending punctuation like "." "?" and "!". However, for New York Times, as the sentences we retrieved are mostly short sentences or part of a sentence, we decided not to count this for New York Times section.

2.3 Total Unique words

We count unique words by making a histogram of all the words using make_hist.prl. We changed the original file to rank the result with a descending order of number in the document.

2.4 Hard Words %

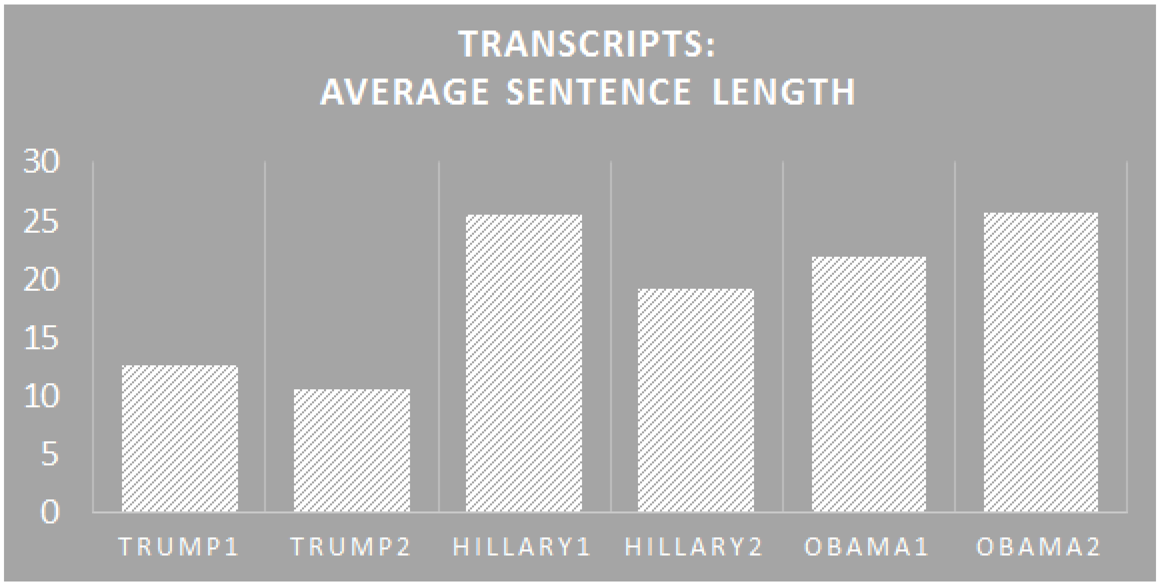
Hard words are defined as words with three or more syllables.



From the figure above, we can see that Mr. Trump uses less hard words in his speech, which people have already noticed.

2.5 Average Sentence length

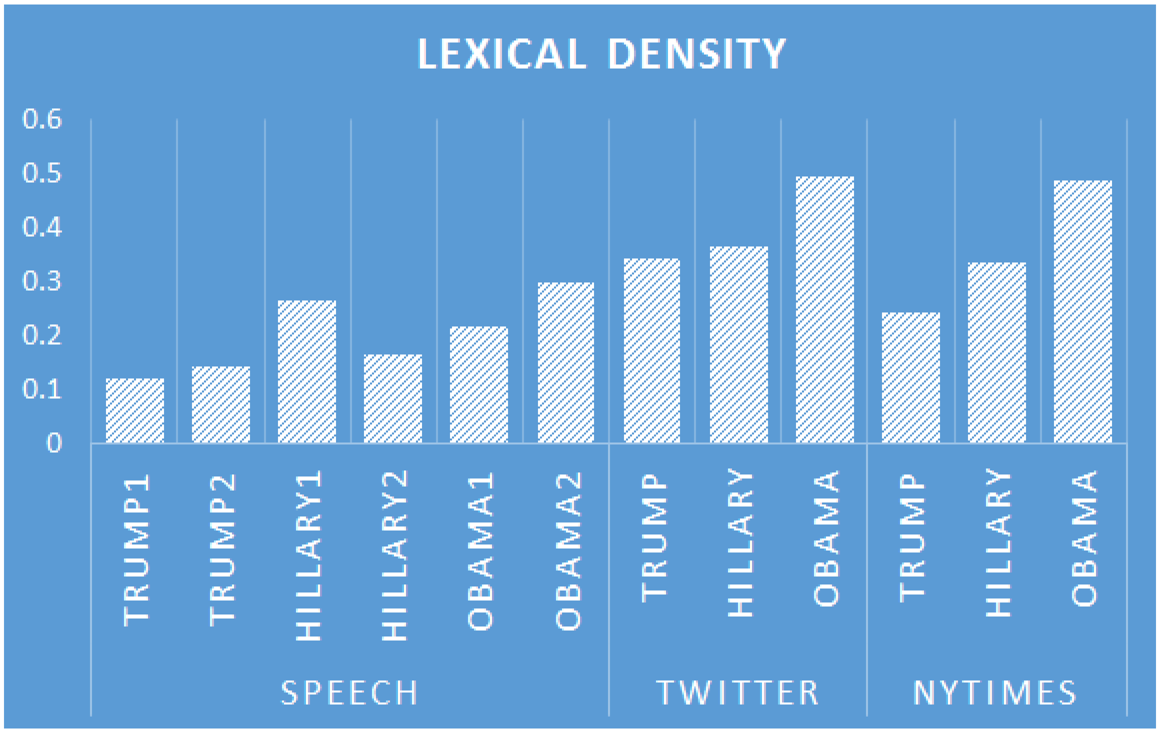
Average sentence length is calculated by Total Words / Total Sentences.



From the figure, we can see that Mr. Trump uses significantly shorter sentences, nearly half as what Obama does. Again, people can tell it when they are listening to Trump's speech.

2.6 LEXICAL DENSITY

Lexical density is a very important index in measuring people's speech, which is a ratio of Unique words/Total words.



In speech, Twitter and New York Times, Trump's lexical density is lower than the other two. Another interesting finding is that the three politicians tend to use more complex sentences in Twitter. The reason why lexical density is high for all the three in NYTimes is because that the reporters tend to pick out those quotes with rich meanings.

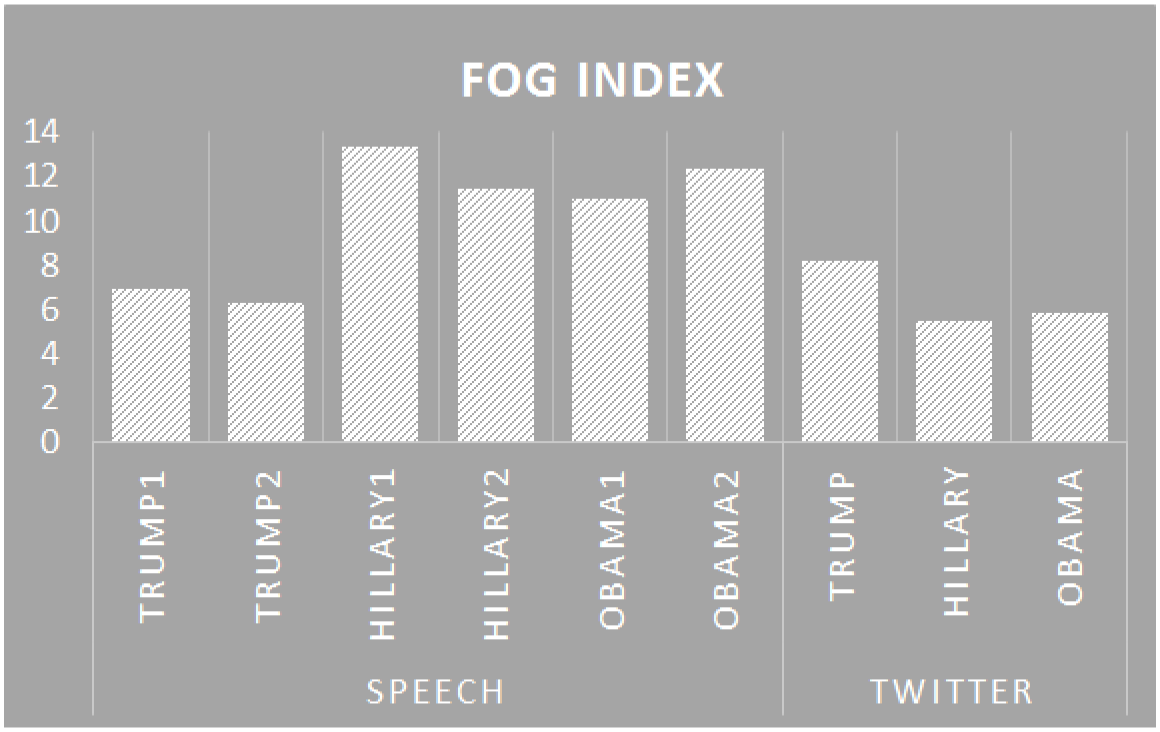
2.7 FOG INDEX

The Fog index measures the readability of a document. It is calculated by (Hard Words%+Average Sentence Length) * 0.4.

FOG INDEX	READING LEVEL BY GRADE
17	College graduate
16	College senior
15	College junior
14	College sophomore
13	College freshman
12	High school senior
11	High school junior
10	High school sophomore
9	High school freshman
8	Eighth grade
7	Seventh grade
6	Sixth grade

Figure 1: https://en.wikipedia.org/wiki/Gunning_fog_index

The fog index is commonly used to confirm that text can be read easily by the intended audience. Texts for a wide audience generally need a fog index less than 12. Texts requiring near-universal understanding generally need an index less than 8.



According to the result, as always, Trump's speech is more readable than the other two. The readability of his speech is between six grade and seventh grade.

However, in terms of Twitter, Trump uses more complicated sentences, as the Fog Index of his Twitter posts exceed the Hillary and Obama by more than 30%. This is probably due to the limit of the number of words in Twitter, where Trump is willing to post slightly difficult things, but the readability is not affected a lot.

2.8 New Dale Chall Formula

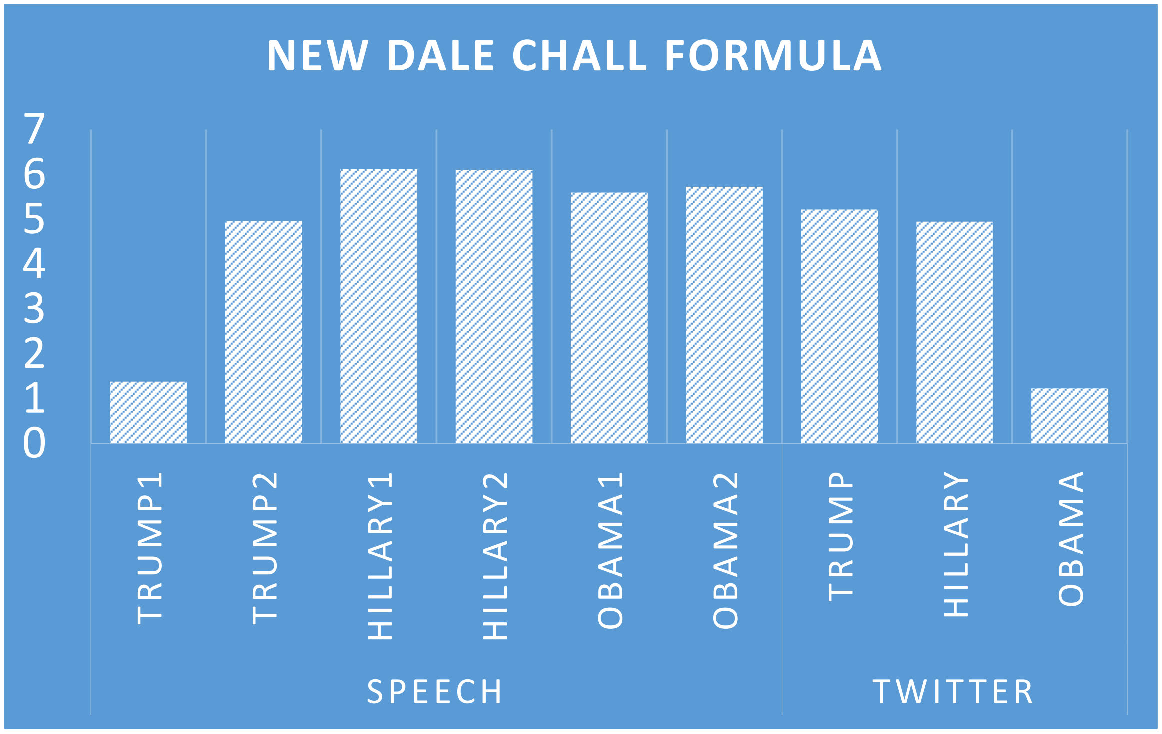
Compute the following equation:
Raw Score = 0.1579 * (PDW) + 0.0496 * ASL
Raw Score = Reading Grade of a reader who can comprehend your text at 3rd grade or below.
PDW = Percentage of Difficult Words
ASL = Average Sentence Length in words
If (PDW) is greater than 5%, then:
Adjusted Score = Raw Score + 3.6365, otherwise Adjusted Score = Raw Score
Adjusted Score = Reading Grade of a reader who can comprehend your text at 4th grade or above.

Step 5: Use the following table to get the Adjusted Grade Level:

ADJUSTED SCORE	GRADE LEVEL
4.9 AND BELOW	Grade 4 and Below
5.0 TO 5.9	Grades 5 - 6
6.0 TO 6.9	Grades 7 - 8
7.0 TO 7.9	Grades 9 - 10
8.0 TO 8.9	Grades 11 - 12
9.0 TO 9.9	Grades 13 - 15 (College)
10 AND ABOVE	Grades 16 and Above (College Graduate)

Figure 2: http://www.readabilityformulas.com/new-dale-chall-readability-formula.php

From the following figure, we can get basically the same conclusion as above. However, Obama's Twitter score is very low using this formula, which could be a result of the low average sentence length and a hard-words ratio lower than 5 that would not be adjusted.



3 EXECUTION DETAILS

3.1 NEW YORK TIMES BOTS

perl robot_trump.pl <logfile> <contentfile> http://www.nytimes.com

perl robot_hillary.pl <logfile> <contentfile> http://www.nytimes.com

```
perl robot_obama.pl <logfile> <contentfile> http://www.nytimes.com
```

These command will generate files

obama.raw hillary.raw and trump.raw

3.2 Twitter Bot

```
perl twitter.pl
```

This will generate files obamatweets.raw, trumptweets.raw and hillarytweets.raw

4 Interesting Observations

Overall more data was extracted for Trump from the New York Times bots than it was extracted for Obama and Hillary. This just shows how much more coverage Trump is getting compared to other candidates.

We paid for 3 weeks' subscription to New York Times so that we could get full access to it instead of just 10 articles.

5 CODE

5.1.1 CODE FOR TWITTER BOT

```
#!/usr/local/bin/perl -w
use strict;
use Carp;
use FileHandle;
use HTTP::Request;
use HTTP::Response;
use HTTP::Status;
use utf8;
use Text::Unidecode;

use Net::Twitter::Lite::WithAPIv1_1;

my $consumer_key = "fo6MDUtBhkZ2MaVB4CFM2mIG1";
my $consumer_secret =
"vIu8ALtsC3HGMPJoaAZWJe7a6Q4Kf1IyGRjZcrbKm2voAUPhnp";
my $token = "731363372084670464-4OOcPt3xHVQHkBsqXz7W9iqO48Sgd9z";
my $token_secret = "I4Cgoni1EQdjtVaTEVGqkc4Ai737W2B49RwHT5LOID6La";

  open(my $trumptweets, '>', 'trumptweets.raw');
  my $st = 'from:realDonaldTrump';
  my $numtweets = 1000;

my $newt = Net::Twitter::Lite::WithAPIv1_1->new(
    consumer_key       => $consumer_key,
   consumer_secret     => $consumer_secret,
    access_token        => $token,
    access_token_secret => $token_secret, ssl=>1);

my $results = $newt->search({q=>$st,count=> $numtweets, lang=>"en"});
for my $status ( @{$results->{statuses}} ) {
    $status->{text} =~ s/([^[:ascii:]]+)/unidecode($1)/ge;
    print $trumptweets "$status->{text}\n\n";
}

close $trumptweets;

open(my $hillarytweets, '>', 'hillarytweets.raw');
  $st = 'from:HillaryClinton';

$results = $newt->search({q=>$st,count=> $numtweets,lang=>"en"});

for my $status ( @{$results->{statuses}} ) {
    $status->{text} =~ s/([^[:ascii:]]+)/unidecode($1)/ge;
    print $hillarytweets "$status->{text}\n\n";
}
close $hillarytweets;

open(my $obamatweets, '>', 'obamatweets.raw');
  $st = 'from:BarackObama';

$results = $newt->search({q=>$st,count=> $numtweets, lang=>"en"});
for my $status ( @{$results->{statuses}} ) {
    $status->{text} =~ s/([^[:ascii:]]+)/unidecode($1)/ge;
```

```
   #$status->{text} =~ s/(.*)http//g;
  #  print $1;
   print $obamatweets "$status->{text}\n\n";
}
close $hillarytweets;



exit (0);
```

5.1.2 CODE for robot_hillary.pl

```
#!/usr/local/bin/perl -w

#
# This program walks through HTML pages, extracting all the links to other
# text/html pages and then walking those links. Basically the robot performs
# a breadth first search through an HTML directory structure.
#
# All other functionality must be implemented
#
# Example:
#
#    robot_base.pl mylogfile.log content.txt http://www.cs.jhu.edu/
#
# Note: you must use a command line argument of http://some.web.address
#       or else the program will fail with error code 404 (document not
#       found).

use strict;
use Carp;
use HTML::LinkExtor;
use HTTP::Request;
use HTTP::Response;
use HTTP::Status;
use LWP::RobotUA;
use URI::URL;

URI::URL::strict( 1 );   # insure that we only traverse well formed URL's

$| = 1;

my $log_file = shift (@ARGV);

my $content_file = shift (@ARGV);

if ((!defined ($log_file)) || (!defined ($content_file))) {
   print STDERR "You must specify a log file, a content file and a base_url\n";
   print STDERR "when running the web robot:\n";
   print STDERR "  ./robot_base.pl mylogfile.log content.txt base_url\n";
   exit (1);
}

open LOG, ">$log_file";
open CONTENT, ">$content_file";

############################################################
```

```
##              PLEASE CHANGE THESE DEFAULTS            ##
############################################################

# I don't want to be flamed by web site administrators for
# the lousy behavior of your robots.

my $ROBOT_NAME = 'LmaoBot/1.0';
my $ROBOT_MAIL = 'lmao5@jhu.edu';

my $robot = new LWP::RobotUA $ROBOT_NAME, $ROBOT_MAIL;

$robot->delay(0.02);

$robot->cookie_jar( {} );

my $base_url    = shift(@ARGV);   # the root URL we will start from
my ($site_name) = $base_url =~ /\.(.+)/;

my @search_urls = ();    # current URL's waiting to be trapsed
my @wanted_urls = ();    # URL's which contain info that we are looking for
my %relevance   = ();    # how relevant is a particular URL to our search
my %pushed      = ();    # URL's which have either been visited or are already
                         #  on the @search_urls array
my %output      = ();

push @search_urls, $base_url;
```

```
while (@search_urls) {

    my $url = shift @search_urls;

    my $parsed_url = eval { new URI::URL $url; };

    next if $@;
    next if $parsed_url->scheme !~/http/i; # case insensitive

    print $parsed_url, "\n";

    print LOG "[HEAD ] $url\n";

    my $request  = new HTTP::Request HEAD => $url;
    my $response = $robot->request( $request );

    # print $response->code, "\n";
    # print $response->content_type, "\n";

    # next if $response->code != RC_OK;

    # next if ! &wanted_content( $response->content_type, $parsed_url );

    print LOG "[GET  ] $url\n";

    $request->method( 'GET' );
    $response = $robot->request( $request );
```

```
if ($response->code == "302") {

    $url = $response->header("Location");

    $request  = new HTTP::Request HEAD => $url;

    $request->method( 'GET' );
    $response = $robot->request( $request );

    # print $response->content;
}

# next if $response->code != RC_OK;

next if $response->content_type !~ m@text/html@;

print LOG "[LINKS] $url\n";

&extract_content ($response->content, $url);

my @related_urls  = &grab_urls( $response->content, $url );

foreach my $link (@related_urls) {

    next if $link =~ /#/;
    next if $link !~ /http/;

    # print $link, "\n";
```

```
            my $full_url = eval { (new URI::URL $link, $response->base)->abs; };

            delete $relevance{ $link } and next if $@;

        next if $full_url =~ /$url#/;

        next if $full_url !~ /$site_name\/(2015|2016)/;

            $relevance{ $full_url } = $relevance{ $link };

            delete $relevance{ $link } if $full_url ne $link;

            push @search_urls, $full_url and $pushed{ $full_url } = 1
               if ! exists $pushed{ $full_url };

    }

    @search_urls =
            sort { -1 * ($relevance{ $a } <=> $relevance{ $b }); } @search_urls;

}

close LOG;
close CONTENT;

exit (0);
```

```
sub extract_content {

    my $content = shift;
    my $url = shift;

    my $words;



    open(my $hillary, '>>', 'hillary.raw');

    while ($content =~
s/Mrs\.\sClinton\s(said|added|claimed|stated|asked|exclaimed)[\s\w]*(\:|\,).\“([\-
\’\.\,\?\!\s\w]+)\”//) {
        $words = $3;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $hillary "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }

    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w\,]*(said|added|claimed|stated|asked|exclaimed)[\s\w]*Mrs\.\
sClinton//) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $hillary "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }



    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w]*Mrs\.\sClinton[\s\w]*(said|added|claimed|stated|asked|excl
aimed)//) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $hillary "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }
    close $hillary;
    return;
}



sub grab_urls {


    my $content = shift;
    my $url = shift;


    my %urls    = ();    # NOTE: this is an associative array so that we only
```

```
                    #       push the same "href" value once.



    skip:


    while ($content =~ s/<\s*[aA]
([^>]*)>\s*(?:<[^>]*>)*(?:([^<]*)(?:<[^aA>]*>)*<\/\s*[aA]\s*>)?//) {


        my $tag_text = $1;
        my $reg_text = $2;


      my $link = "";
      my $weight = 0;


      # print $tag_text, "\n";
      # print $reg_text, "\n";


        if (defined $reg_text) {


          $reg_text =~ s/[\n\r]/ /;
          $reg_text =~ s/\s{2,}/ /;


        if ($reg_text =~ /hillary|clinton|election|democratic|primary|politics/i) {
          # print "OK", $reg_text, "\n";
          $weight ++;
        }
          #
          # compute some relevancy function here
          #
        }

        if ($tag_text =~ /href\s*=\s*(?:["']([^"']*)["']|([^\s])*)/i) {

        $link = $1 || $2;
        $link = "" if (!defined $link);
        # print $link, "\n";

        # next if $link !~ /$site_name/;

        if ($link =~ /election|primary|democratic|politics/i) {
          $weight ++;
        }



        if ($link =~ /hillary|clinton/i) {
          $weight += 2;
        }


          #
          # okay, the same link may occur more than once in a
          # document, but currently I only consider the last
          # instance of a particular link
          #
        # print $weight, "\n";
          $relevance{ $link } = $weight;
          $urls{ $link }      = 1;
        }
```

```
        # print $reg_text, "\n" if defined $reg_text;
        # print $link, "\n\n";
  }

  return keys %urls;   # the keys of the associative array hold all the
                       # links we've found (no repeats).
}
```

5.1.3 CODE for robot_trump.pl

```
#!/usr/local/bin/perl -w

#
# This program walks through HTML pages, extracting all the links to other
# text/html pages and then walking those links. Basically the robot performs
# a breadth first search through an HTML directory structure.
#
# All other functionality must be implemented
#
# Example:
#
#    robot_base.pl mylogfile.log content.txt http://www.cs.jhu.edu/
#
# Note: you must use a command line argument of http://some.web.address
#       or else the program will fail with error code 404 (document not
#       found).
```

```
use strict;

use Carp;
use HTML::LinkExtor;
use HTTP::Request;
use HTTP::Response;
use HTTP::Status;
use LWP::RobotUA;
use URI::URL;

URI::URL::strict( 1 );   # insure that we only traverse well formed URL's

$| = 1;

my $log_file = shift (@ARGV);

my $content_file = shift (@ARGV);

if ((!defined ($log_file)) || (!defined ($content_file))) {
   print STDERR "You must specify a log file, a content file and a base_url\n";
   print STDERR "when running the web robot:\n";
   print STDERR "  ./robot_base.pl mylogfile.log content.txt base_url\n";
   exit (1);
}

open LOG, ">$log_file";
open CONTENT, ">$content_file";
```

```
############################################################
##              PLEASE CHANGE THESE DEFAULTS             ##
############################################################

# I don't want to be flamed by web site administrators for
# the lousy behavior of your robots.

my $ROBOT_NAME = 'LmaoBot/1.0';
my $ROBOT_MAIL = 'lmao5@jhu.edu';

my $robot = new LWP::RobotUA $ROBOT_NAME, $ROBOT_MAIL;

$robot->delay(0.02);

$robot->cookie_jar( {} );

my $base_url    = shift(@ARGV);   # the root URL we will start from
my ($site_name) = $base_url =~ /\.(.+)/;

my @search_urls = ();    # current URL's waiting to be trapsed
my @wanted_urls = ();    # URL's which contain info that we are looking for
my %relevance   = ();    # how relevant is a particular URL to our search
my %pushed      = ();    # URL's which have either been visited or are already
                         #  on the @search_urls array
my %output      = ();

push @search_urls, $base_url;
```

```
while (@search_urls) {

    my $url = shift @search_urls;

    my $parsed_url = eval { new URI::URL $url; };

    next if $@;
    next if $parsed_url->scheme !~/http/i; # case insensitive

    print $parsed_url, "\n";

    print LOG "[HEAD ] $url\n";

    my $request  = new HTTP::Request HEAD => $url;
    my $response = $robot->request( $request );

    # print $response->code, "\n";
    # print $response->content_type, "\n";

    # next if $response->code != RC_OK;

    # next if ! &wanted_content( $response->content_type, $parsed_url );

    print LOG "[GET  ] $url\n";

    $request->method( 'GET' );
    $response = $robot->request( $request );
```

```
if ($response->code == "302") {

    $url = $response->header("Location");

    $request  = new HTTP::Request HEAD => $url;

    $request->method( 'GET' );
    $response = $robot->request( $request );

    # print $response->content;
}

# next if $response->code != RC_OK;

next if $response->content_type !~ m@text/html@;

print LOG "[LINKS] $url\n";

&extract_content ($response->content, $url);

my @related_urls  = &grab_urls( $response->content, $url );

foreach my $link (@related_urls) {

    next if $link =~ /#/;
    next if $link !~ /http/;

    # print $link, "\n";
```

```
            my $full_url = eval { (new URI::URL $link, $response->base)->abs; };

            delete $relevance{ $link } and next if $@;

        next if $full_url =~ /$url#/;

        next if $full_url !~ /$site_name\/(2015|2016)/;

            $relevance{ $full_url } = $relevance{ $link };

            delete $relevance{ $link } if $full_url ne $link;

            push @search_urls, $full_url and $pushed{ $full_url } = 1
                if ! exists $pushed{ $full_url };

    }

    @search_urls =
            sort { -1 * ($relevance{ $a } <=> $relevance{ $b }); } @search_urls;

}

close LOG;
close CONTENT;

exit (0);
```

```
sub extract_content {

    my $content = shift;
    my $url = shift;

    my $words;

    # print $content;
     open(my $trump, '>>', 'trump.raw');

    while ($content =~
s/Trump\s(said|added|claimed|stated|asked|exclaimed)[\s\w]*(\:|\,).\“([\-
\’\.\,\?\!\s\w]+)\”//) {
        $words = $3;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $trump "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }

    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w\,]*(said|added|claimed|stated|asked|exclaimed)[\s\w(Mr\.)]*
Trump//) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $trump "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }



    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w(Mr\.)]*Trump[\s\w]*(said|added|claimed|stated|asked|exclai
med)//) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $trump "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }
    close $trump;
    return;
}



sub grab_urls {

    my $content = shift;
    my $url = shift;

    my %urls    = ();    # NOTE: this is an associative array so that we only
                         #       push the same "href" value once.
```

```
   skip:

   while ($content =~ s/<\s*[aA]
([^>]*)>\s*(?:<[^>]*>)*(?:([^<]*)(?:<[^aA>]*>)*<\/\s*[aA]\s*>)?//) {

         my $tag_text = $1;
         my $reg_text = $2;

      my $link = "";
      my $weight = 0;

      # print $tag_text, "\n";
      # print $reg_text, "\n";

         if (defined $reg_text) {

            $reg_text =~ s/[\n\r]/ /;
            $reg_text =~ s/\s{2,}/ /;

         if ($reg_text =~ /trump|election|republican|primary|politics/i) {
            # print "OK", $reg_text, "\n";
            $weight ++;
         }
            #
            # compute some relevancy function here
            #
         }
```

```
 if ($tag_text =~ /href\s*=\s*(?:["']([^"']*)["']|([^\s])*)/i) {

$link = $1 || $2;
$link = "" if (!defined $link);
# print $link, "\n";

# next if $link !~ /$site_name/;

if ($link =~ /election|republican|primary|politics/i) {
   $weight ++;
}



if ($link =~ /trump/i) {
   $weight += 2;
}


    #
    # okay, the same link may occur more than once in a
    # document, but currently I only consider the last
    # instance of a particular link
    #
# print $weight, "\n";
    $relevance{ $link } = $weight;
    $urls{ $link }      = 1;
 }
```

```
        # print $reg_text, "\n" if defined $reg_text;
        # print $link, "\n\n";
    }

    return keys %urls;   # the keys of the associative array hold all the
                         # links we've found (no repeats).
}
```

5.1.4 CODE for robot_obama.pl

```
#!/usr/local/bin/perl -w


#
# This program walks through HTML pages, extracting all the links to other
# text/html pages and then walking those links. Basically the robot performs
# a breadth first search through an HTML directory structure.
#
# All other functionality must be implemented
#
# Example:
#
#    robot_base.pl mylogfile.log content.txt http://www.cs.jhu.edu/
#
# Note: you must use a command line argument of http://some.web.address
#       or else the program will fail with error code 404 (document not
#       found).

use strict;
use Carp;
use HTML::LinkExtor;
use HTTP::Request;
use HTTP::Response;
use HTTP::Status;
use LWP::RobotUA;
use URI::URL;

URI::URL::strict( 1 );   # insure that we only traverse well formed URL's

$| = 1;

my $log_file = shift (@ARGV);

my $content_file = shift (@ARGV);

if ((!defined ($log_file)) || (!defined ($content_file))) {
   print STDERR "You must specify a log file, a content file and a base_url\n";
   print STDERR "when running the web robot:\n";
   print STDERR "  ./robot_base.pl mylogfile.log content.txt base_url\n";
   exit (1);
}

open LOG, ">$log_file";
open CONTENT, ">$content_file";

############################################################
##            PLEASE CHANGE THESE DEFAULTS           ##
```

```
############################################################

# I don't want to be flamed by web site administrators for
# the lousy behavior of your robots.

my $ROBOT_NAME = 'LmaoBot/1.0';
my $ROBOT_MAIL = 'lmao5@jhu.edu';

my $robot = new LWP::RobotUA $ROBOT_NAME, $ROBOT_MAIL;

$robot->delay(0.02);

$robot->cookie_jar( {} );

my $base_url    = shift(@ARGV);   # the root URL we will start from
my ($site_name) = $base_url =~ /\.(.+)/;

my @search_urls = ();    # current URL's waiting to be trapsed
my @wanted_urls = ();    # URL's which contain info that we are looking for
my %relevance   = ();    # how relevant is a particular URL to our search
my %pushed      = ();    # URL's which have either been visited or are already
                         #  on the @search_urls array
my %output      = ();

push @search_urls, $base_url;

while (@search_urls) {
```

```
my $url = shift @search_urls;

my $parsed_url = eval { new URI::URL $url; };

next if $@;
next if $parsed_url->scheme !~/http/i; # case insensitive

print $parsed_url, "\n";

print LOG "[HEAD ] $url\n";

my $request  = new HTTP::Request HEAD => $url;
my $response = $robot->request( $request );

# print $response->code, "\n";
# print $response->content_type, "\n";

# next if $response->code != RC_OK;

# next if ! &wanted_content( $response->content_type, $parsed_url );

print LOG "[GET  ] $url\n";

$request->method( 'GET' );
$response = $robot->request( $request );

if ($response->code == "302") {
    $url = $response->header("Location");

    $request  = new HTTP::Request HEAD => $url;

    $request->method( 'GET' );
    $response = $robot->request( $request );

    # print $response->content;
}

# next if $response->code != RC_OK;

next if $response->content_type !~ m@text/html@;

print LOG "[LINKS] $url\n";

&extract_content ($response->content, $url);

my @related_urls  = &grab_urls( $response->content, $url );

foreach my $link (@related_urls) {

    next if $link =~ /#/;
    next if $link !~ /http/;

    # print $link, "\n";

        my $full_url = eval { (new URI::URL $link, $response->base)->abs; };
```

```
            delete $relevance{ $link } and next if $@;

        next if $full_url =~ /$url#/;

        next if $full_url !~ /$site_name\/(2015|2016)/;

            $relevance{ $full_url } = $relevance{ $link };

            delete $relevance{ $link } if $full_url ne $link;

            push @search_urls, $full_url and $pushed{ $full_url } = 1
               if ! exists $pushed{ $full_url };

    }

    @search_urls =
            sort { -1 * ($relevance{ $a } <=> $relevance{ $b }); } @search_urls;

}

close LOG;
close CONTENT;

exit (0);


sub extract_content {
    my $content = shift;
    my $url = shift;

    my $words;

    # print $content;

    open(my $obama, '>>', 'obama.raw');

    while ($content =~
s/Obama\s(said|added|claimed|stated|asked|exclaimed)[\s\w]*(\:|\,).\“([\-
\’\.\,\?\!\s\w]+)\”//) {
        $words = $3;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $obama "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }

    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w\,]*(said|added|claimed|stated|asked|exclaimed)[\s\w]*Obam
a//) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $obama "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }

    while ($content =~ s/\“([\-
\’\.\,\?\!\s\w]+)\”[\s\w]*Obama[\s\w]*(said|added|claimed|stated|asked|exclaimed)/
/) {
        $words = $1;
        last if defined($output{$words});
        # print $trump "\n";
        print "$words \n";
        print $obama "$words \n";
        # print $trump"\n";
        $output{$words} = $url;
    }
    close $obama;

    return;
}


sub grab_urls {

    my $content = shift;
    my $url = shift;

    my %urls    = ();    # NOTE: this is an associative array so that we only
                         #       push the same "href" value once.
```

```
   skip:

   while ($content =~ s/<\s*[aA]
([^>]*)>\s*(?:<[^>]*>)*(?:([^<]*)(?:<[^aA>]*>)*<\/\s*[aA]\s*>)?//) {

          my $tag_text = $1;
          my $reg_text = $2;

      my $link = "";
      my $weight = 0;

      # print $tag_text, "\n";
      # print $reg_text, "\n";

          if (defined $reg_text) {

              $reg_text =~ s/[\n\r]/ /;
              $reg_text =~ s/\s{2,}/ /;

          if ($reg_text =~ /obama|politics/i) {
              # print "OK", $reg_text, "\n";
              $weight ++;
          }
              #
              # compute some relevancy function here
              #
          }
```

```
if ($tag_text =~ /href\s*=\s*(?:["']([^"']*)["']|([^\s])*)/i) {

$link = $1 || $2;
$link = "" if (!defined $link);
# print $link, "\n";

# next if $link !~ /$site_name/;

if ($link =~ /democratic|politics|obamacare/i) {
   $weight ++;
}

if ($link =~ /obama/i) {
   $weight += 2;
}

    #
    # okay, the same link may occur more than once in a
    # document, but currently I only consider the last
    # instance of a particular link
    #
# print $weight, "\n";
    $relevance{ $link } = $weight;
    $urls{ $link }      = 1;
 }

 # print $reg_text, "\n" if defined $reg_text;
 # print $link, "\n\n";
```

```
  }

  return keys %urls;   # the keys of the associative array hold all the
                       # links we've found (no repeats).
}
```

5.1.5 CODE for make_hist.prl

```
#!/usr/bin/perl -w

##########################################################
##########################################################
my $totdocfreq = 0;

while (<STDIN>) {
  chop;
  if ($_ =~ /^[.]I/) {
    if ($totdocfreq > 0) {
      &process_doc_freqs;
    }
  }
  else {
                        $_ = lc $_;
    $curdocfreq{$_}++;
    $totdocfreq++;
  }
}

if ($totdocfreq > 0) {
   &process_doc_freqs;
}

&print_freqs;

############################################################
############################################################

sub process_doc_freqs {

  while (($term,$freq) = each %curdocfreq) {
    $corpusfreq{$term}+=$freq;
  }
  $totdocfreq = 0;
  undef %curdocfreq;
}

sub print_freqs {

            foreach my $term (sort { -1 * ($corpusfreq{$a} <=> $corpusfreq{$b}) }
keys %corpusfreq) {
    printf("%s\t%s\n", $term, $corpusfreq{$term} );
            }

}
```

5.1.6 CODE for stemAndToke.sh

```
#!/usr/bin/bash
```

```
perl -p -i -e 's/\s/\n/g' nytimes/trump
perl -p -i -e 's/\s/\n/g' nytimes/trump.stemmed
./make_hist.prl < nytimes/trump.stemmed > nytimes/trump.hist
```