Mining the Web for Data, Part II

This class is a continuation of the previous class that was taught by John Perry and Aron Pilhofer.

The simplest sites to scrape are ones where you can simply go to a URL, download the page, and parse the HTML using regular expressions, just as you learned in the last class. But as you do more and more web scraping, you'll eventually find sites where this process of just changing values in the URL to fetch pages no longer works. And thought the regular expression is a powerful tool, there are better ways to find information on Web pages.

Some sites might require you to fill out a search form first, some might require a username and password, some might set cookies on your browser that need to be passed from page-to-page, and some others still might require you to go through pages in a specified order before displaying results.

In the first part of this class today, we are going to scrape the IRE investigative reporter directory by following links, and then extracting information from Web tables. In the second part, we will learn how to fill out a Web search form to grab information from an online search form.

To do this, we're going to learn how to use Perl modules, chunks of code written by someone else to solve a similar problem and then shared with the Perl community through CPAN, the Comprehensive Perl Archive Network.

The first module we're going to use, which is already installed when you download Perl 5.10 from ActiveState is WWW::Mechanize, which emulates a Web browser that you can use to navigate from page-to-page, fill out forms and click "submit" buttons just like a human would.

We'll go look at the IRE Directory of Investigative Journalists site now. Its URL is:

```
http://bolles.ire.org/dij/
```

Script 1: Find the page

To start our script, we need to tell the Perl program which modules and pragmas we are going to use. In this case, we want to use the strict pragma which will give us better error messages and warnings of typos in our program, and the WWW::Mechanize module, which is the module that does all the hard word of automating our search for the Web page we want. So, we start our program off with:

```
#!/bin/perl

use strict;
use WWW::Mechanize;
```

Next, we'll define a couple of variables. The first is the directory where we are going to save all of our downloaded files. (If this directory doesn't exist, go create it now.) An important thing to note is that when we use directory paths in Perl, you either need to use a forward slash to separate the directories ("`c:/temp/crime`") – or use two backward slashes ("`c:\\temp\\crime`"). It will **not** work if you only use one backward slash between directory parts ("`c:\temp\crime`"). This has to do with Perl's heritage on Unix-based systems, which unlike Microsoft, use forward slashes as the directory separator.

We also define a variable that contains the URL for the page containing the search form.

```
my $output_file = 'c:/temp/ire/car_people.csv';
 my $starting_url = 'http://bolles.ire.org/dij/';
```

Next, we'll create a virtual Web browser in our Perl program that we'll name `$ua`, for user agent. (Technically, this is called an object of the WWW::Mechanize class, but since we don't want to get into all the details of object-oriented programming in this class, we'll just refer to it as the browser throughout the rest of the examples.)

```
my $ua = WWW::Mechanize->new();
```

We start by telling the browser to go fetch a Web page. From now on, you'll notice that whenever we want to tell the browser to do something, we'll use `$ua`, followed by that funny arrow, and what we want it to do. (How do you know what the browser can do? The documentation for the WWW::Mechanize module lists all the commands – also known as methods – available for use. The URL for the documentation is at the end of this document.)

```
$ua->get( $starting_url );
```

So, now we've sent the Perl program off fetch our starting page, we need to fill in the blanks on the form. We can test to see that we successfully fetched the page by simply looking at it – by using www-mech's content method.

```
print $ua->content;
```

Assuming that worked, we'll comment out that line, telling Perl to skip it now when it runs the script, and move on.

```
#print $ua->content;
```

This being NICAR, we're only really interested in CAR experts. So from our start page, we'll follow the link that says: "click here to get an index of interests." The www-mech follow_link method can find the link you want in several ways, by looking for the link text or the url, or by counting down the order in which it appears in the page code. We're going to use the link text:

```
$ua->follow_link( text => 'click here to get an index of interests.' );
```

We arrive at a page that lists the interests of all the reporters, with links that lead to lists of reporters. We want to follow the CAR link. This time, we'll use the URL from the link's href. But instead of giving www-mech the full URL, we'll use a regular expression to match a part of the link. The URLs all end with a search term, "interest=", then the specific interest. To find CAR reporters, we need the link that ends with "interest=CAR".

```
$ua->follow_link( url_regex => 'interest=CAR' );
```

Script 1: Scrape the page

Now that we've arrive safely at our destination, it's time to scrape the data. You see a list of reporters with their names, affiliation, address, phone and interest (all CAR, in this case).

If you look at the page's underlying code, you find that, even though you can't see the borders, each reporter is enclosed by an HTML table. Each table has rows with two cells, one on the right with the date, and one on the left with the name of the data element. This is going to make the page easy to scrape, with the help of a second module we will add to our script: HTML::TableExtract:

```
use HTML::TableExtract;
```

When we add this line to our code, however, Komodo likely underlines the statement in red and complains that it can't find the module. That's because, unlike www-mech, TableExtract is not a core module that comes installed with Perl 5.10. We have to install it.

Open the Windows command prompt window and type this command:

```
ppm install HTML::TableExtract
```

When you hit return, the Perl package manager will find the module in a repository on the internet, download it, install it and eventually report its success. For 99 percent of Perl modules, this is all that is involved with installation.

Now that we have our new module, we'll use it to pull the table data out of the mess of HTML. But first, as with www-mech, we have to create out TableExtract object and store it in a variable we'll call `$te` for table extract:

```
my $te = HTML::TableExtract->new;
```

And then we'll use our object to parse the HTML, which we'll get through www-mech's content method:

```
$te->parse( $ua->content );
```

Our `$te` object now has all of the page's tables, with all the rows and cells and all the information they enclose. To get to that information, we'll create two `for` loops, which step one by one through a collection of things we give it. The outer loop with step through all the tables. For each table, an inner loop with step through each row to grab our data, which we'll save in a hash.

```
for my $table ( $te->tables ) {
   for my $row ( $table->rows ) {
        . . .
   }
}
```

We're going to save the row information in a hash, which we'll have to create outside of the rows loop. Inside the rows loop, we can grab the data from the $row object using the order number of the cell, starting with zero. The data element name, `$row->[0]`, will become the hash key. The data in `$row->[1]` will become the hash value:

```
for my $table ( $te->tables ) {
   my %data;
   for my $row ( $table->rows ) {
        $data{ $row->[0] } = $row->[1];
   }
```

```
}
```

After we've looped through all the rows and have our data stored nicely in our hash, we can print it to a text file as a comma-separated record. So we'll need to open a file for writing and add a print statement. And when we're all through, we'll close our file:

```
open OUT, ">$output_dir";
for my $table ( $te->tables ) {
   my %data;
   for my $row ( $table->rows ) {
        $data{ $row->[0] } = $row->[1];
   }
   print OUT $data{ 'Name:' },         ',',
             $data{ 'Affiliation:' }, ',',
             $data{ 'Address:' },     ',',
             $data{ 'Phone:' },       ',',
             $data{ 'Interest:' },    "\n";
}
close OUT
```

Script 2: Forms

Now we know how to grab information from HTML tables. But what if getting to the data involves more than following links? To get to the data, you often have to do a search using a web for. Again, www-mech comes to the rescue.

Go to the following link:

```
http://www.perryhome.net/nicar
```

You'll find a search page where you can enter either a state abbreviation or a name to search for people interested in CAR. The result of the search, conveniently, looks exactly like the results on NICAR page. So all we have to change is how we get to the data.

If you look at the source code, you see that the page actually includes two separate forms, one named 'byState' and another named 'byName'. To fill out one of the forms and submit the search, first we have to tell our www-mech browser which form we want to use.

We want to search by state, and there are two ways to pick the proper form – by it's order in the source code:

```
$ua->form_number( 1 )
```

Or by its name:

```
$ua->form_name( 'byState' )
```

We'll use the form name.

Next, we'll need to do a little bit of detective work to determine what the form element where we enter the state abbreviation is called.

The HTML tag for a text input is <INPUT>. In the source, we can see what the inputs are named. Reading through the HTML, we find this: `<input type="text" name="state" />.` Thus, we tell www-mech to fill out the "`state`" field with a value of, since we're in Texas, "`TX`".

```
$ua->set_field( state => 'TX' );
```

Now the only thing left is to click the button. And since the form has only one button, www-mech can figure out what it is we want to click all by itself:

```
$ua->click;
```

Now that the form is submitted, we can test our efforts by referring to `$ua->content`. And if that's satisfactory, we already have the code to grab the information off our results page and print it to a file. So this is where our script is now:

```
#!/usr/bin/perl
use strict;
use Data::Dump qw( dump );

use WWW::Mechanize;
use HTML::TableExtract;

my $output_file = 'c:/temp/ire/car_people.csv';
my $starting_url = 'http://www.perryhome.net/nicar';

my $ua = WWW::Mechanize->new;
$ua->get( $starting_url );

#$ua->follow_link( text => 'click here to get an index of interests.' );
#$ua->follow_link( url_regex => qr/interest=CAR/ );
open OUT, ">$output_dir";

$ua->form_name( 'byState' );
$ua->set_fields( state => 'TX' );
$ua->click;

my $te = HTML::TableExtract->new;
$te->parse( $ua->content );

for my $table ( $te->tables ) {
    my %data;
    for my $row ( $table->rows ) {
        $data{ $row->[0] } = $row->[1];
    }

    print OUT $data{ 'Name:' }, ",",
              $data{ 'Affiliation:' }, ",",
              $data{ 'Address:' }, ",",
              $data{ 'Phone:' }, ",",
              $data{ 'Interests:' }, "\n";
}
```

```
close OUT
```

Program 3. Going back for more

But what if our interests are less parochial than just Texas CAR people. To get information about reporters in other states, or even in every state, we'll have to go back to the original search page. Again, www-mech makes our life easy with the `back` method:

```
$ua->back;
```

But first, to keep track of where we want to go, we'll create an array of all the state abbreviations we're interested in:

```
$my @states = ( 'TX','OK','DC','OK' );
```

Then we'll use a `for` loop again to step through all the states:

```
#!/usr/bin/perl
use strict;
use Data::Dump qw( dump );

use WWW::Mechanize;
use HTML::TableExtract;

my $output_file = 'c:/temp/ire/car_people.csv';
 my $starting_url = 'http://www.perryhome.net/nicar/';

my $ua = WWW::Mechanize->new;
$ua->get( $starting_url );

open OUT, ">$output_dir";
 my @states = ( 'TX','OK','DC','OK' );

for my $state ( @states ) {
   $ua->form_name( 'byState' );
   $ua->set_fields( state => $state );
   $ua->click;

   my $te = HTML::TableExtract->new;
   $te->parse( $ua->content );

   for my $table ( $te->tables ) {
       my %data;
       for my $row ( $table->rows ) {
           $data{ $row->[0] } = $row->[1];
       }

     print OUT $data{ 'Name:' }, ",",
               $data{ 'Affiliation:' }, ",",
               $data{ 'Address:' }, ",",
```

```
            $data{ 'Phone:' }, ",",
            $data{ 'Interests:' }, "\n";
   }
   $ua->back
}
close OUT
```

Full listings of programs

Program 1

```
#!/usr/bin/perl
use strict;
use Data::Dump qw( dump );

use WWW::Mechanize;
use HTML::TableExtract;

my $output_file = ‘c:/temp/ire/car_people.csv’;
my $starting_url = ‘http://bolles.ire.org/dij/’;

my $ua = WWW::Mechanize->new;
$ua->get( $starting_url );

$ua->follow_link( text => 'click here to get an index of interests.' );
$ua->follow_link( url_regex => qr/interest=CAR/ );
open OUT, “>$output_file”;

my $te = HTML::TableExtract->new;
$te->parse( $ua->content );

for my $table ( $te->tables ) {
    my %data;
    for my $row ( $table->rows ) {
        $data{ $row->[0] } = $row->[1];
    }

    print OUT $data{ 'Name:' }, ",",
              $data{ 'Affiliation:' }, ",",
              $data{ 'Address:' }, ",",
              $data{ 'Phone:' }, ",",
              $data{ 'Interests:' }, "\n";
}
close OUT
```

Program 2

```
#!/usr/bin/perl
use strict;
use Data::Dump qw( dump );

use WWW::Mechanize;
use HTML::TableExtract;

my $output_file = ‘c:/temp/ire/car_people.csv’;
my $starting_url = ‘http://www.perryhome.net/nicar’;

my $ua = WWW::Mechanize->new;
$ua->get( $starting_url );

open OUT, “>$output_file”;

$ua->form_name( 'byState' );
$ua->set_fields( state => 'TX' );
$ua->click;

my $te = HTML::TableExtract->new;
$te->parse( $ua->content );

for my $table ( $te->tables ) {
    my %data;
    for my $row ( $table->rows ) {
        $data{ $row->[0] } = $row->[1];
    }

    print OUT $data{ 'Name:' }, ",",
              $data{ 'Affiliation:' }, ",",
              $data{ 'Address:' }, ",",
              $data{ 'Phone:' }, ",",
              $data{ 'Interests:' }, "\n";
}
close OUT
```

Program 3

```
#!/usr/bin/perl
use strict;
use Data::Dump qw( dump );

use WWW::Mechanize;
use HTML::TableExtract;

my $output_file = 'c:/temp/ire/car_people.csv';
my $starting_url = 'http://www.perryhome.net/nicar';

my $ua = WWW::Mechanize->new;
$ua->get( $starting_url );

open OUT, ">$output_file";
my @states = ( 'TX','OK','DC','OK' );

for my $state ( @states ) {
    $ua->form_name( 'byState' );
    $ua->set_fields( state => $state );
    $ua->click;

    my $te = HTML::TableExtract->new;
    $te->parse( $ua->content );

    for my $table ( $te->tables ) {
        my %data;
        for my $row ( $table->rows ) {
            $data{ $row->[0] } = $row->[1];
        }

        print OUT $data{ 'Name:' }, ",",
                  $data{ 'Affiliation:' }, ",",
                  $data{ 'Address:' }, ",",
                  $data{ 'Phone:' }, ",",
                  $data{ 'Interests:' }, "\n";
    }
    $ua->back
}
close OUT
```