Component and Interface Design

Jackie Li

Emoveo exists as a local perl script that can be ran by the user using Windows Command Prompt or Terminal. Emoveo is a simple script with no dependencies other than the specific modules that is specified by the program requirements in the design document. The goal is to provide enough automation and user control such that no other overhead costs other than the requirements are necessary for the users.

```
use strict;
use warnings;
use Lingua::EN::StopWords qw(%StopWords);
use Lingua::Stem::En;
use Data::Dump qw(dump);
use Path::Class;
use Lingua::EN::Fathom;
use utf8;
use File::Slurp qw(read_file write_file);
use PDL;
use LWP::Simple;
use HTML::TreeBuilder;
use HTML::FormatText;
use Locale::Country;
use Locale::Language;
use Lingua::Concordance;
use Lingua::EN::Ngram;
use Encode qw(decode encode);
use Text::TermExtract;
use RTF::Writer;
use WWW::Wikipedia;
use Lingua::EN::Tagger;
use Tkx;
use List::Compare;
```

The initial invocation of the script begins the program described by the workflow section in the functional specifications document.

The user is then tasked with inputting a document (plain text file) for Emoveo to read. Once read, the script peforms and initial analysis of the input file to determine the size of the document and what are the fastest methods to redact the document. For texts less than 5000 words, Emoveo allows the user to do paragraph redactions while for text with greater than 5000 words, Emoveo gives the user the option of manually inputting words to remove from the document delimited by commas. This primary analysis is dependent on the Lingua::EN::Fathom module.

```
my $text = new Lingua::EN::Fathom;
$text->analyse_file($inputfile);
my $accumulate = 1;
$text->analyse_block( my $text_string, $accumulate );

my $num_chars             = $text->num_chars;
my $num_words             = $text->num_words;
my $percent_complex_words = $text->percent_complex_words;
my $num_sentences         = $text->num_sentences;
my $num_text_lines        = $text->num_text_lines;
my $num_blank_lines       = $text->num_blank_lines;
my $num_paragraphs        = $text->num_paragraphs;
my $syllables_per_word    = $text->syllables_per_word;
my $words_per_sentence    = $text->words_per_sentence;

my $fog     = $text->fog;
my $flesch  = $text->flesch;
my $kincaid = $text->kincaid;

print( $text->report );
print "\n\n";
system('pause');
```

Fathom uses whitespace, word length and punctuation markers to determine the number of characters, words, complex words, sentences, lines, and paragraphs. It also generates a Fog, Flesch, and Kincaid readability index of the text.

The next element that affects the paragraph redaction as well as the keyword tree build is the module Text::TermExtract. The module uses word length, frequency, and an exclusion list to extract interesting keywords from the text. The exclusion list removes stop words such as the, and, etc. which do not provide much semantic meaning to the text. A list of maximum 5 keywords are generated for the paragraph redaction and a list of maximum 20 keywords are generated for the user to iterate and edit.

```
use Text::TermExtract;
use File::Slurp;

$file = "C:/Perl/STR.txt";
$text = read_file($file);

$ext = Text::TermExtract->new();

for my $word($ext->terms_extract($text,{max =>20})){
        print "$word\n";
}
```

```
my $ext = Text::TermExtract->new();
$ext->exclude(
    [
        'said',         'a',            'able',         'about',
        'above',        'abst',         'accordance',   'according',
        'accordingly',  'across',       'act',          'actually',
        'added',        'adj',          'affected',     'affecting',
        'affects',      'after',        'afterwards',   'again',
        'against',      'ah',           'all',          'almost',
        'alone',        'along',        'already',      'also',
        'although',     'always',       'am',           'among',
        'amongst',      'an',           'and',          'announce',
        'another',      'any',          'anybody',      'anyhow',
        'anymore',      'anyone',       'anything',     'anyway',
        'anyways',      'anywhere',     'apparently',   'approximately',
        'are',          'aren',         'arent',        'arise',
        'around',       'as',           'aside',        'ask',
        'asking',       'at',           'auth',         'available',
        'away',         'awfully',      'b',            'back',
        'be',           'became',       'because',      'become',
```

The keywords are pushed into an array while the exclusion keywords are pushed into an array reference to store the data. Each iteration over the keyword array puts the keywords through the WWW::Wikipedia module to perform a search on the Wikipedia website. Then, the content is extracted and ran through Text::TermExtract to create a set of 20 keywords associated with the keyword searched in an independent array.

```
for ( my $counters = 0 ; $counters < $#initial_keys ; $counters++ ) {
    my $key_to_search = $initial_keys[$counters];
    my $wiki          = WWW::Wikipedia->new();
    my $result        = $wiki->search($key_to_search);
    my $filewiki      = "C:/Perl/wikistore.txt";
    open( WIKI, ">", $filewiki ) or die;
    if ( $result !~ /^$/ ) {
        if ( $result->text() ) {
            print WIKI $result->text();
            print $result->text();
            close WIKI;

            #my $service = read_file($filewiki);
            my $wikifile = read_file($filewiki);
            my $wikiext  = Text::TermExtract->new();
            $wikiext->exclude( [ 'refer', 'ref' ] );
            for my $keyword (
                $wikiext->terms_extract( $wikifile, { max => 20 } ) )
            {
                print $keyword. "\n";
                if ( $counters == 0 ) {
                    push @keys_for_0, $keyword;
                }
```

Once the user decides to use the keywords to search the document, Lingua::Concordance with uses a custom radius integer dependent on the word count in the document to search for a phrase

containing the keyword and surround words dependent on the radius. The user can then choose how much to redact. A simple implementation of this is used in conjunction with Lingua::EN::Tagger to search the file for nouns and noun phrases (maximum 5 word phrases) to run back into Lingua::Concordance.

```
        if ( $finder =~ /$initial_keys[0]/ ) {
            my $concordance0 = Lingua::Concordance->new;
            $concordance0->radius($integer);
            my $reader0 = read_file($loopreplacefile);
            $concordance0->text($reader0);
            $concordance0->query(
                $initial_keys[0], $keys_for_0[0],  $keys_for_0[1],
                $keys_for_0[2],   $keys_for_0[3],  $keys_for_0[4],
                $keys_for_0[5],   $keys_for_0[6],  $keys_for_0[7],
                $keys_for_0[8],   $keys_for_0[9],  $keys_for_0[10],
                $keys_for_0[11],  $keys_for_0[12], $keys_for_0[13],
                $keys_for_0[14],  $keys_for_0[15], $keys_for_0[16],
                $keys_for_0[17],  $keys_for_0[18], $keys_for_0[19]
            );
            foreach ( $concordance0->lines ) {
                print "$_\n";
                if ( $_ =~ /~~/ ) {
                    print "Would you like to keep this line?\t";
    elsif ( $finder =~ /clear all proper/i ) {
        my $properfile = "C:/Perl/proper.txt";
        open( PROP, ">", $properfile );
        my $parserforproper = new Lingua::EN::Tagger;
        my $parserfile      = read_file($loopreplacefile);
        my $tagged_text     = $parserforproper->add_tags($parserfile);

        #print $tagged_text;
        my %wordlist = $parserforproper->get_words($parserfile);
        my @wordlistq;
        foreach my $parsedkey ( keys %wordlist ) {

            #print $parsedkey;
            $parsedkey =~ s/\\\w+//;
            push @wordlistq, $parsedkey;
        }
        print PROP "$_\n" for @wordlistq;
        close PROP;
```

Lastly, the automated redaction uses simple regex in order to remove elements. Locale::Country and Locale::Language are lists with country names and language names that are iterated over by a for loop to help speed up redaction. The plain text modified is then printed into a rtf using RTF::Writer.

The last major module List::Compare compare the initial keyword list with a keyword list generated by the redacted text to give the shared keywords and the entire combined keyword list to give the user an idea of what information was redacted.

```
my $lc = List::Compare->new( \@initial_keys, \@final_keys );

my @intersection = $lc->get_intersection;

print "Keywords that intersect (are in initial keys and final keys)\n";
print "$_\n" for @intersection;

my @union = $lc->get_union;

print "Keywords that are union (are in initial keys or final keys)\n";
print "$_\n" for @union;
```