```
=comment

    Programming Assignment 4: Decision Lists
    Evan Oman
    11/10/2014

    Usage:

    perl decision-list.pl **Training Data**.txt **Test Data**.txt **Decision List**.txt > **Answers**.txt

    Description of Problem:

    The goal of this lab is to create a decision list for word sense disambiguation. We are focusing on the specific ambiguity:


            WordNet Case 15: telephone line, phone line, telephone circuit, subscriber line, line---(a telephone connection)
           /
    line =>
           \
            WordNet Case 22: line, product line, line of products, line of merchandise, business line, line of business -- (a particular kind of product or merchandise; "a nice line of shoes")


    The method used to disambiguate these cases is outlined in David Yarowsky's paper: "DECISION LISTS FOR LEXICAL AMBIGUITY RESOLUTION: Application to Accent Restoration in Spanish and French." There the author uses decision lists in order to properly apply accents in Spanish and French texts but many of the methods outlined are directly applicable to our problem.

    We begin by analyzing the given training data using several collocational factors to determine how context correlates to word sense. Once we have recorded counts of how many times a certain factor is associated with each word sense, we sort all of the factors by their log-likelihood defined below:

                              P(Sense_1 | Collocation_i)
    log likelihood = Abs(Log( -------------------------))
                              P(Sense_2 | Collocation_i)

    This equation essentially tells us how correlated some Collocation_i is to one of the two word senses. For example, if there is no correlation, then the P(Accent_Pattern_1 | Collocation_i) and P(Accent_Pattern_2 | Collocation_i) should be pretty close to .5, thus we are taking the log of 1 which is 0. However if some Collocation_i is almost always associated with Accent_Pattern_1, then the log likelihood would be very large.

    Once we have this list of collocational factors sorted by their correlation with one of the two senses we can label word senses in some testing data set. For each testing sentence, we iterate through the decision list continuing until some collocational factor applies to the testing sentence and then apply the corresponding word sense to the testing sentence.

=cut

use DecisionList;

main();

sub main
{
    chdir "line-data";    #Moves us to the correct directory
    local $/;            #Enables localized slurp mode

    #Grabs the input params
    my $trainFile = $ARGV[0];
    my $testFile = $ARGV[1];
```

```
49      my $outputFile = $ARGV[2];
50
51      #Employs our decision list class which learns from the given data and generates a decision list
52      $decisionList = new DecisionList($trainFile);
53
54      #Here we apply the decision list generated about to the specified test data set
55      $decisionList->classifyTestData($testFile);
56
57      $decisionList->printDecisionTree($outputFile);
58  }
```

```
1  =comment
2      This is my decision list class which contains the structure of the decision list along with all
   of the necessary utility functions.
3  =cut
4
5
6  package DecisionList;
7  use Switch;
8  use CollocationFactor;
9
10 #Here I define some enums which will serve a purpose throughout this class
11 use constant PREV         =>    1;
12 use constant SURR        =>    2;
13 use constant NEXT        =>    3;
14 use constant PREV_1        =>    4;
15 use constant NEXT_1        =>    5;
16 use constant SENSE_1     =>    "phone";
17 use constant SENSE_2     =>    "product";
18 use constant NIL         =>    -1;
19
20 #This is the initialer for the class
21 sub new
22 {
23     my $class = shift;
24     my $self = {
25         _trainingFileName        =>    shift,
26         _decisionList            =>    [],
27         _collocationData        =>    [],
28     };
29
30 =comment
31     Note that the structure of the collocatrionData hashes is:
32     hash = {
33         "w1/w2" => {
34                 sense1 => count
35                 sense2 => count
36         },
37     .
38     .
39     .
40     }
41 =cut
42
43     bless $self, $class;
44
45     #Parses the training data and populates the appropriate hashes
46     $self->populateCollocationData($self->{_trainingFileName});
47
48     #Now that we have all of the data collected, we can make our decision list and sort the factors
   according to their log likelihood
49     $self->createDecisionList();
50
51     return $self;
52 }
53
54 =comment
55     Here is where we will parse the training set and populate the collocation data hashes. Based off
   of the Yarkowsky's paper I have decided to include the following collocational factors:
56         -prev: Previous two words    (-2,-1 W)
57         -surr: Surrounding words    (-1,+1 W)
58         -next: Next two words        (+1,+2 W)
59 =cut
60 sub populateCollocationData
61 {
```

```
62      my( $self ) = @_;
63      my $filename = $_[1];
64
65      open(MYFILE, $filename);
66      my $lines = <MYFILE>;
67      close(MYFILE);
68
69      #Sets the topic
70      $_ = $lines;
71
72      #Translates everything to lower case
73      tr/[A-Z]/[a-z]/;
74
75      #Some useful variables
76      my $sense;
77      my $context;
78      my @collocationArr = [];
79
80      for my $type (PREV..NEXT_1)
81      {
82          push(@collocationArr, {});
83      }
84
85      #Based on the format of the given training data, we can loop through each "instance" which
    contains a hand tagged word sense along with context.
86      #A note on the regex: +? gives a non-greedy match so we can go one tagset at a time rather than
    the overall <instance ... </instance> match
87      for (m/<instance.+?<\/instance>/gms)
88      {
89          #Grabs the word sense in this instance
90          if (m/senseid=\"(\w+)\"/gms)
91          {
92              $sense = $1;
93              $self->{_totalOccurancesHash}{$sense}++;
94          }
95          else
96          {
97              print "PARSE ERROR, EXITING";
98              return;
99          }
100
101         #Grabs the context sentence[s?]
102         if (m/<context>(.+)<\/context>/gms)
103         {
104             $context = $1;
105
106             #Cleans things up a bit
107             $context =~ tr/([\,\"])/ $1 /;
108
109             #Here we grab all of the words in a +/-2 window of the word line, again using the fact
    that the data is structured such that the instance of line that is of interest is wrapped in a <head>
    </head>
110             if ($context =~ m/\s+([^\s]+)\s+([^\s]+)\s+<head>line[s]?<\/head>\s+([^\s]+)\s+
    ([^\s]+)/gms    )
111             {
112                 #At this point we have:
113                 #    -(-2) word => $1
114                 #    -(-1) word => $2
115                 #    -(+1) word => $3
116                 #    -(+2) word => $4
117
118                 $collocationArr[PREV]{"$1\/$2"}{$sense}++;
119                 $collocationArr[SURR]{"$2\/$3"}{$sense}++;
120                 $collocationArr[NEXT]{"$3\/$4"}{$sense}++;
121                 $collocationArr[PREV_1]{"$2"}{$sense}++;
```

```
122                 $collocationArr[NEXT_1]{"$3"}{$sense}++;
123             }
124             else
125             {
126                 print "PARSE ERROR, EXITING";
127                 return;
128             }
129         }
130     }
131     #Outputs the data array to the contex data object
132     @{$self->{_collocationData}} = @collocationArr;
133 }
134
135 =comment
136     Takes the collected collocation data and creates an array of collocation decision factor objects which are sorted by their log likelihood as a measure of the efficacy.
137 =cut
138 sub createDecisionList
139 {
140     #Parse the data of all three collocation types and push the outgoing list into the master decision list
141     my( $self ) = @_;
142     my @decisionArr = ();
143
144     #For each type, parse the collocation data
145     for my $type (PREV..NEXT_1)
146     {
147         $self->parseCollocationData($type);
148     }
149
150     #Now we sort the master decision list by the log score of collocation factor objects
151     @{$self->{_decisionList}} = sort {abs($b->getLogScore()) <=> abs($a->getLogScore())} @{$self->{_decisionList}};
152 }
153
154 =comment
155     For the specified collocation type, add the factors to the global decision list
156 =cut
157 sub parseCollocationData
158 {
159     my($self, $type) = @_;
160     my %data = %{$self->{_collocationData}[$type]};
161     my @factorArr = ();
162
163     #First loop through
164     for my $words (keys %data)
165     {
166         #Here I perform a simple add one interpolation
167         $data{$words}{${\SENSE_1}}++;
168         $data{$words}{${\SENSE_2}}++;
169
170         my $total = $data{$words}{${\SENSE_1}} + $data{$words}{${\SENSE_2}};
171
172         #Now we can compute the log likelihood score:
173         my $pNum = $data{$words}{${\SENSE_1}}/$total;    #Numerator probability
174         my $pDen = $data{$words}{${\SENSE_2}}/$total;    #Denominator probability
175
176         my $logScore = log($pNum/$pDen);
177
178         #The sign of the log score tells us which probability was greater. If the sign is positive then the the log term was greater than 1 and thus the numerator was greater. A similar argument applies to the denominator. Thus I have implemented the below ternary that carries out the logic specified above
179         my $sense = $logScore > 0 ? ${\SENSE_1} : ($logScore < 0 ? ${\SENSE_2} : ${\NIL});
180
```

```
181         my $factor = new CollocationFactor($type, $words, $logScore, $sense);
182 
183         push(@{$self->{_decisionList}}, $factor);
184     }
185 }
186 
187 =comment
188     Apply this instance's decision list to the given test data
189 =cut
190 sub classifyTestData
191 {
192     my($self, $testFileName) = @_;
193 
194     open(MYFILE, $testFileName);
195     my $lines = <MYFILE>;
196     close(MYFILE);
197 
198     $_ = $lines;
199 
200 
201     #Loop over each given instance
202     for (m/<instance.+?<\/instance>/gms)
203     {
204         $instanceID = "";
205         $answerWordSense = "";
206 
207         #Grabs the instance ID
208         if (m/id=\"(.+?)\"/gms)
209         {
210             $instanceID = $1;
211         }
212 
213         #Grabs the context sentence[s?]
214         if (m/<context>(.+)<\/context>/gms)
215         {
216             $context = $1;
217 
218             #Cleans things up a bit
219             $context =~ tr/([\,\"])/ $1 /;
220 
221             #Here we grab all of the words in a +/-2 window of the word line, again using the fact
    that the data is structured such that the instance of line that is of interest is wrapped in a <head>
    </head>
222             if ($context =~ m/\s+([^\s]+)\s+([^\s]+)\s+<head>line[s]?<\/head>\s+([^\s]+)\s+
    ([^\s]+)/gms    )
223             {
224                 #At this point we have:
225                 #    -(-2) word => $1
226                 #    -(-1) word => $2
227                 #    -(+1) word => $3
228                 #    -(+2) word => $4
229 
230                 $answerWordSense = $self->applyDecisionList($1, $2, $3, $4);
231             }
232         }
233         #If we made it here, then one of the factors must have applied. Thus we can simply write the
    answer to the standard I/O buffer, as directed
234         print "<answer instance=\"$instanceID\" senseid=\"$answerWordSense\"\/>\n";
235     }
236 }
237 
238 =comment
239     Actually loops through the sorted decision list and applies the most likely tag
240 =cut
241 sub applyDecisionList
```

```
242 {
243     my($self, $one, $two, $three, $four) = @_;
244     #Now we cycle through all of the possible factors, starting with the factors that had the highest
    log likelihood
245     for my $factor (@{$self->{_decisionList}})
246     {
247         #We must handle each type of factor differently
248 
249         my $type = $factor->getType();
250         my $words = $factor->getWords();
251         my $wordSense = $factor->getWordSense();
252         my $logLikelihood = $factor->getLogScore();
253 
254         if ($logLikelihood == 0)
255         {
256             #APPLY NIL TAG
257             $answerWordSense = -1;
258 
259             #exit the loop, we found our tag
260             return;
261         }
262 
263         #Handle the current factor based on its type
264         switch ($type)
265         {
266             case PREV
267             {
268                 #Must match previous 2:
269                 my @wordsArr = split "\/", $words;
270                 if ($wordsArr[0] eq $one && $wordsArr[1] eq $two)
271                 {
272                     #APPLY TAG
273                     #exit the loop, we found our tag
274                     return $wordSense;
275                 }
276             }
277             case SURR
278             {
279                 #Must match surrounding 2:
280                 my @wordsArr = split "\/", $words;
281                 if ($wordsArr[0] eq $two && $wordsArr[1] eq $three)
282                 {
283                     #APPLY TAG
284                     #exit the loop, we found our tag
285                     return $wordSense;
286                 }
287             }
288             case NEXT
289             {
290                 #Must match the next 2:
291                 my @wordsArr = split "\/", $words;
292                 if ($wordsArr[0] eq $three && $wordsArr[1] eq $four)
293                 {
294                     #APPLY TAG
295                     #exit the loop, we found our tag
296                     return $wordSense;
297                 }
298             }
299             case PREV_1
300             {
301                 #Must match previous 1:
302                 if ($words eq $two)
303                 {
304                     #APPLY TAG
305                     #exit the loop, we found our tag
```

```
306                     return $wordSense;
307                 }
308             }
309             case NEXT_1
310             {
311                 #Must match next 1:
312                 if ($words eq $three)
313                 {
314                     #APPLY TAG
315                     #exit the loop, we found our tag
316                     return $wordSense;
317                 }
318             }
319             else
320             {
321                 #APPLY TAG
322                 #exit the loop, we found our tag
323                 return -1;
324             }
325 
326         }
327     }
328 }
329 
330 =comment
331     Outputs the decision list in a human readable manner
332 =cut
333 sub printDecisionTree
334 {
335     my($self, $filename) = @_;
336     open(my $fh, '>', $filename);
337 
338     my $counter = 0;
339     for my $factor (@{$self->{_decisionList}})
340     {
341         print $fh "--------------------\n\n"
342         print $fh "Log score: $factor->getLogScore()\n";
343         my $words = $factor->getWords();
344         my $wordSense = $factor->getWordSense();
345         #Handle the current factor based on its type
346         switch ($factor->getType())
347         {
348             case PREV
349             {
350                 print $fh "If the previous two words are $words, then the word sense is
    $wordSense\n";
351             }
352             case SURR
353             {
354                 print $fh "If the surrounding two words are $words, then the word sense is
    $wordSense\n";
355             }
356             case NEXT
357             {
358                 print $fh "If the next two words are $words, then the word sense is $wordSense\n";
359             }
360             case PREV_1
361             {
362                 print $fh "If the previous word is $words, then the word sense is $wordSense\n";
363             }
364             case NEXT_1
365             {
366                 print $fh "If the next word is $words, then the word sense is $wordSense\n";
367             }
368         }
```

```
369         print $fh "--------------------\n\n"
370     }
371     close $fh;
372
373 }
374
375 #Boiler plate class file ending
376 1;
```

```
1  =comment
2      This class contains the structure of a single collocational factor. There are three possible types
   of collocation factors that I haver chosen to implement:
3          1. prev: The previous 2 words
4          2. surr: The surrounding 2 words
5          3. next: The next 2 words
6  =cut
7
8
9
10 #Unofficial enum for the collocation types:
11 #$prev = 1;
12 #$surr = 2;
13 #$next = 3;
14 package CollocationFactor;
15 sub new
16 {
17     my $class = shift;
18     my $self = {
19         _type        =>    shift,
20         _words        =>    shift,        #Contains the collocation data for the (-2,-1 W) case
21         _logScore    =>    shift,        #Container for the decision list
22         _wordSense    =>    shift,        #The winning word sense, could be NIL(-1)
23     };
24
25     bless $self, $class;
26     return $self;
27 }
28
29
30 sub getType
31 {
32     my( $self ) = @_;
33     return $self->{_type};
34 }
35
36 sub getWords
37 {
38     my( $self ) = @_;
39     return $self->{_words};
40 }
41
42 sub getLogScore
43 {
44     my( $self ) = @_;
45     return $self->{_logScore};
46 }
47
48
49 sub getWordSense
50 {
51     my( $self ) = @_;
52     return $self->{_wordSense};
53 }
54
55 1;
```

```
1  =comment
2      The goal of this script is to compare the output of decision-tree.pl with the key file.
3
4      Usage: perl scorer.pl **TEST FILE** **KEY FILE**
5  =cut
6
7  #We assume here that the lines are in one to one correspondence(which they are)
8  main();
9
10 sub main
11 {
12     #Grabs the input params
13     my $testFile = $ARGV[0];
14     my $keyFile = $ARGV[1];
15
16     #Get all of the senseids from the test file:
17     @testAnswers = ();
18     open(MYFILE, $testFile);
19     my $lines = <MYFILE>;
20     $_ = $lines;
21     for (<MYFILE>)
22     {
23         if (m/senseid=(".*")/gms)
24         {
25             push @testAnswers, $1;
26         }
27     }
28     close(MYFILE);
29
30     #Get all of the senseids from the test file:
31     @keyAnswers = ();
32     open(MYFILE, $keyFile);
33     $lines = <MYFILE>;
34     $_ = $lines;
35     for (<MYFILE>)
36     {
37         if (m/senseid=(".*")/gms)
38         {
39             push @keyAnswers, $1;
40         }
41     }
42     close(MYFILE);
43
44     my $right;
45     my $total;
46     my $arrSize = @testAnswers;
47
48     for (my $i = 0; $i < $arrSize; $i++)
49     {
50         if ($testAnswers[$i] eq $keyAnswers[$i])
51         {
52             $right++;
53         }
54         $total++;
55     }
56
57     my $accuracy = $right/$total;
58
59     print "This decision tree got $right correct out of $total total giving an overall accuracy of
   $accuracy%.";
60 }
```

```
***This is some sample output from my decision list program, the first 2 pages***

--------------------

Log score: 3.49650756146648
If the previous word is telephone, then the word sense is phone

--------------------

Log score: -3.3499040872746
If the next word is of, then the word sense is product

--------------------

Log score: 3.09104245335832
If the previous two words are on/the, then the word sense is phone

--------------------

Log score: -2.99573227355399
If the surrounding two words are a/of, then the word sense is product

--------------------

Log score: 2.77258872223978
If the previous word is access, then the word sense is phone

--------------------

Log score: 2.39789527279837
If the previous two words are of/the, then the word sense is phone

--------------------

Log score: 2.39789527279837
If the surrounding two words are the/was, then the word sense is phone

--------------------

Log score: -2.39789527279837
If the previous word is a, then the word sense is product

--------------------

Log score: -2.30258509299405
If the surrounding two words are new/of, then the word sense is product

--------------------

Log score: -2.30258509299405
If the previous word is car, then the word sense is product

--------------------

Log score: -1.94591014905531
If the previous two words are a/new, then the word sense is product

--------------------

Log score: -1.94591014905531
If the previous word is ps/2, then the word sense is product

--------------------

```

```
Log score: -1.94591014905531
If the previous word is computer, then the word sense is product

--------------------

Log score: 1.94591014905531
If the next two words are was/dead, then the word sense is phone

--------------------

Log score: 1.87180217690159
If the next word is was, then the word sense is phone

--------------------

Log score: 1.79175946922806
If the surrounding two words are the/., then the word sense is phone

--------------------

Log score: 1.79175946922806
If the surrounding two words are the/1, then the word sense is phone

--------------------

Log score: 1.79175946922806
If the surrounding two words are telephone/and, then the word sense is phone

--------------------

Log score: 1.7730673362159
If the previous word is the, then the word sense is phone

--------------------

Log score: -1.6094379124341
If the previous two words are introduced/a, then the word sense is product

--------------------

Log score: 1.6094379124341
If the surrounding two words are direct/to, then the word sense is phone

--------------------

Log score: 1.6094379124341
If the surrounding two words are telephone/1, then the word sense is phone

--------------------

Log score: 1.6094379124341
If the surrounding two words are access/., then the word sense is phone

--------------------

Log score: 1.6094379124341
If the surrounding two words are telephone/., then the word sense is phone

--------------------

Log score: 1.6094379124341
If the next two words are to/the, then the word sense is phone

--------------------

```