Interpreting CLD2 Unit Test Output
Dick Sites 2013.06.04

Compile the CLD2 unit test via

```
g++ -O2 -m64  cld2_unittest.cc   cldutil.cc cldutil_shared.cc compact_lang_det.cc
compact_lang_det_hint_code.cc   compact_lang_det_impl.cc  debug.cc
fixunicodevalue.cc   generated_entities.cc  generated_language.cc
generated_ulscript.cc    getonescriptspan.cc lang_script.cc offsetmap.cc
scoreonescriptspan.cc   tote.cc utf8statetable.cc
cld_generated_cjk_uni_prop_80.cc cld2_generated_cjk_compatible.cc
cld_generated_cjk_delta_bi_32.cc generated_distinct_bi_0.cc
cld2_generated_quadchrome0527.cc cld2_generated_deltaoctachrome0527.cc
cld2_generated_distinctoctachrome0527.cc
cld_generated_score_quad_octa_1024_256.cc    -o cld2_unittest
```

If you then run

```
cld2_unittest --html 2>CLD2UnitTestOutput.html
```

you will get an HTML file summarizing the language detection lookups. Open the existing one in your browser and follow along.

The first line says file = cld2_unittest. After that, there are multiple groups of lines describing each detection done. The sample text to detect is in five big groups. First is a single string of English, as a simple test case. Then there are strings in the 19 languages that CLD2 detects solely based on their Unicode script. Next is a group of four languages detected by looking at single letters (unigrams). Following that are the 60 languages detected by looking at groups of four letters (quadgrams). Finally, there are a few miscellaneous detection strings.

For each string to be detected, the output has one or more lines of labelled text, a **DocTote** line, and a **final result** line. The text shown has been preprocessed to be just the letters and marks of the original text, converted to lowercase. All HTML tags, digits, and punctuation are stripped out. Chunks of about 40 to 120 bytes are individually scored, to allow intermixed text in multiple languages. Each chunk scored is shown with the most likely language code given in square brackets: "[en]" for example signifying English. If the detected language is the same as the last chunk, the brackets are empty: "[]". ISO language codes are used throughout. After the brackets, the scored text is shown with text color and background color varying based on the language. English is black text on light yellow.

In the first string, two chunks are scored, both detected as English.

The cryptic DocTote line shows all the languages detected in the input string or document, each as a group of five items: subscript, language, bytes, probability score, and reliability score. Ignore the subscript. The byte count is the sum of the chunk lengths that were detected as that language. The probability score is the sum of the log probabilities for letter combinations in that language; larger is better. The reliability score will be 100 times the byte count for text deemed

100% reliable, and less than that otherwise.

The final result line gives up to three most likely language codes, reliability score 0..100, and percent of total text in that language. These are followed by a total lowercase-text-bytes count an equal sign and the name of summary language. The summary is often the top language, but can be the second language if the top one is English, French, German, Italian, or Spanish and the second one is a substantial portion of the document. You probably want to pay more attention to the top language. The byte count here always includes one extra blank off the end of the text.

In the first English example, the total text is 254 bytes, all English. The English text percentage 100 * 253/244 is truncated toward zero to give 99%.

The next 20 examples are script-only languages in alphabetical order, Armenian..Thai. Not much interesting happens here.

The next four examples are the CJK languages, detected via lookups on single letters, Chinese..Korean. Simplified and Traditional Chinese are distinguished. Again, not much interesting happens here.

The next 60 examples are the languages detected via quadgrams, again in alphabetical order, Afrikaans..Yiddish. We will highlight the interesting wrinkles.

Albanian (sq) has a slightly lower reliability of 96%. Its probability score is actually a little too high for good Albanian text, so its reliability is shaved a little.

Arabic shows the limits of very short text, with only 39 bytes total. The notation "[ar*.32/fa.10]" shows two unreliably-close close top languages, Arabic (ar) and Persian (fa). The asterisk indicates unreliable results throughout. The 32 and 10 are the raw log probability scores for this chunk. The unreliability comes from the shortage of text, less than about 60 bytes for a reasonable chunk.

Bihari is the first of our "problem children" examples; it is statically close to Hindi when looking at groups of four letters. Of the six chunks scored, the first looks more like Hindi, while the other five look more like Bihari. Overall, the DocTote shows 200 bytes of Hindi and 1002 bytes of Bihari. For close language pairs, CLD2 merges the less-likely one into the more likely one, shown here as "{CloseLangPair: hi.100R,200B => bh}". The final result labels all 1203 bytes Bihari.

Catalan shows a dip in reliability, because many of the letter combinations could also be French or Spanish.

Cebuano has one chunk out of four that could well be Tagalog instead. In this case, the probability score for Tagalog is too low compared to real Tagalog text, so that chunk is marked

unreliable, leaving just 75% of the text as Cebuano and 25% unknown.

Croatian is another close pair, scoring very close to Serbian-Latin. The slightly larger number of bytes in Croatian win out.

Czech reliability dips a bit because some letter combinations could also be Slovak. Similarly, Danish dips a bit because some letter combinations could also be Norwegian. And French dips a bit because some letter combinations could be Italian.

Indonesian and Malay are statically our closest pair, with almost half the letter combinations in common. So two of the five chunks of Indonesian look a bit more like Malay, but these are merged at the end into Indonesian. The later Malay text all scores as Malay, but usually there would be some Indonesian chunks intermixed.

Marathi text actually has some Telugu, Hindi, and Urdu mixed in, making 9 chunks total. The Urdu is so short and unreliable that it is dropped, along with the Hindi. But the Telegu remains, accounting for 3% of the total text.

Nepali reliability is low because the letter combinations look a lot like Hindi also.

Romanian is detected in both the Latin and Cyrillic scripts. (Tagalog is detected in both the Latin and Baybayin scripts.)

Serbian-Cyrillic is straightforward to distinguish, but Serbian-Latin often looks a lot like Croatian.

Swahili has one chunk that looks a bit like (kinya)Rwanda.

After Yiddish, the final group of four input strings show a few other features.

The two Indonesian and Malay examples are nearly-identical Wikipedia page beginnings, from an article on Japanese sukiyaki (the food). They put stress on CLD2's ability to distinguish these close languages.

The next example is intermixed French and English text, with runs in each language a little too short to be easy. The initial detection scores five chunks, but you can see that the boundaries do not line up very well with the actual language boundaries, and do not even line up with word boundaries sometimes. There is a fundamental bootstrapping problem in detecting mixtures of languages. CLD2 scores relatively short but arbitrary chunks to detect whether there is more than one language. It helps if the individual runs in each language are 1.5 chunks or more in length. Once multiple languages have been detected, there is an optional refinement pass to look for better boundaries. Doing so approximately doubles the detection time. Often it is sufficient just to have rough percentages-of-total-text for each language, so that is the default, shown here, roughly 60/40% English/French. The summary says French to highlight the substantial amount

of the secondary language.

If you rerun with the --vector flag, CLD2 will return a vector of language spans to its caller. While forming this vector, it also tries to sharpen the boundaries between differing languages. In this example, it does a little better, but is not perfect.

The very last example shows the Easter Egg string "qpdbmrmxyzptlkuuddlrlrbas" derived from pretty descenders/ascenders, Superman, and Konami. These letter patterns do not occur in any training text, and they artificially map to one of 23 less-used languages based on the build date of the quadgram lookup table. This can be used to see exactly which of two or three different versions you might be using. The string is repeated and intermixed with a little other text to defeat CLD2's mechanisms that ignore short text and ignore repeated text. For a build date of June 4, 2013, the mapping is to Swahili.

The last line of the unittest output is PASS or FAIL.

Even more detailed output, first English example

If you run

```
cld2_unittest --html --cr --verbose --vector 2>CLD2UnitTestOutputVerbose.html
```

you will get an HTML file exposing the full detail of CLD2's inner workings. There are 11 steps shown, nine of which are new -- DocTote and Final result carry over from the discussion above. The rest are discussed in order below.

1. ScoreOneScriptSpan
2. DumpHitBuffer
3. DumpLinearBuffer
4. DumpChunkStart
5. ScoreOneChunk
6. DumpSummaryBuffer
7. SharpenBoundaries
8. DumpSummaryBuffer
9. DocTote
10. Final result
11. DumpResultChunkVector

ScoreOneScriptSpan shows a single span of same-script preprocessed text to be scored. A large document or one with many scripts will show and score multiple ScriptSpans. The script, byte count and apostrophe-delimited string is shown. The first English text ScriptSpan "ScoreOneScriptSpan(Latn,254)" shows 254 bytes of text in the Latin script.

DumpHitBuffer The first step in scoring a (quadgram) ScriptSpan is to look up all the groups of four letters, then groups of eight letters at beginnings of words for a delta-correction and also for

distinctive words across sets of very-close languages. The distinctive lookup also does pairs of words. Not all lookups hit in the scoring tables. The ones that do are put into the HitBuffer, in three columns labelled Q (Quadgrams) L (deLta) and D (Distinctive). For each hit, the byte offset within ScriptSpan is given, the internal table subscript, and a few bytes of the looked-up text (length does not match lookup length). Once the hit buffer is filled, the actual text is not used further in scoring.

The first English text hit "Q[0]6,5760,scatio" shows a quadgram hit at offset 6 on the four letters "scat". Earlier letter combinations "_conf" and "nfis", where underscore indicates the beginning or end of a word, did not hit. (The letters "_conf" are not used because they occur with nearly-equal probabilities in Portuguese, Latin, French, Interlingua, Italian, Romansch, and Romanian. The letters "nfis" are not used because they occur with nearly-equal probabilities in Romanian, Latin, Estonian, Latvian, German, Albanian, and Esperanto.)

The first delta hit "L[0]85,186,members" shows a hit at offset 85 on the letters "_members_". Delta refers to the idea that when this word is scored as quadgrams _memb/mber/rs_ the result is different than the top language for the entire word. The delta scoring adds enough to bring the proper languages to the top, Luxembourgish and English in this case. The delta table is relatively small because it only contains frequent words whose top language differs from the underlying quadgram result.

The first and only distinctive hit "D[0]209,3,importan" shows a hit at offset 209 on "_importan". The distinctive table only contains words or word pairs that separate languages in close sets, ignoring all other languages. In this case, _importan boosts Spanish over Portuguese.

DumpLinearBuffer The Hit Buffer is filled by scanning the text twice, once for quadgrams and once for octagrams. But it turns out that for the rest of the scoring, these need to be sorted by offset. Rather than do something fancy, the HitBuffer entries are simply copied over in order to the LinearBuffer. During this copying, quadgram hits are counted and every 20 arbitrarily mark the beginning of a new scoring chunk. The number 20 is a balance between noisier statistics for fewer hits and bigger mixed language runs for more hits.

DumpChunkStart Shows the LinearBuffer subscript for the beginning of each chunk; three chunks for the English example.

ScoreOneChunk shows the blow-by-blow language probability scoring for each hit. Each letter combination maps to a packed group of up to three languages and log probabilities. The log probabilities are quantized over a range of 1/2**4.8 to 1/2**24.0 in increments of 2**1.6, with smaller values reflecting less-frequent occurrence in the given language. On this scale, 0 corresponds to 1/2**24.0 and 12 to 1/2**4.8 (about 1/28) The quadgram hit "Q:_conf=en.1" shows just one language, English and a low probability of roughly 1/2**22.4. Two entries later, the quadgram hit "Q:tion_=en.6~fr.5~da.4" shows modest probabilities for the word ending "tion_" in English, French, and Danish, with English most likely.

In the second chunk, "_importan" gives a distinct boost to Spanish (vs. the other close languages Portuguese and Galician). The last few distinct boosts encountered are carried into subsequent chunks to give them a bias toward the more likely of close languages. We see this in the "Distinct_boost: es.4" notation after the second chunk. We see this in action a bit in the Bihari example, distinguishing from Hindi. Distinct boosts are particularly important for distinguishing Croatian/Serbian and Indonesian/Malay.

If there were language hints supplied by the caller of CLD2 or by embedded HTML/XML `lang="..."` tags, they would be added in and shown in the "LangPrior_boost:" notation.

The scoring just adds up the language log probabilities (effectively multiplying the original probabilities) within each chunk and declares the largest sum the most likely language for that chunk. In the first English chunk, these totals are shown on the line

```
  en.134 fr.19 84B 20# Latn 100Rd 100Rs
```

a total score of 134 for English and 19 for French over 84 bytes and 20 hits of preprocessed Latn text. Four-letter ISO script codes are used throughout. The reliability measure "100Rd" signifies that the difference in the top two scores is large enough to be deemed 100% reliable. The reliability measure "100Rs" signifies that the absolute score of 134 for 84 bytes of English is reasonably close to the expected score per 1000 bytes for English text.

DumpSummaryBuffer shows the individual scores for each chunk, with a dummy chunk at the end just to hold the offset of the first unused text byte and the subscript of the first unused LinearBuffer entry. Normally, these scores and byte sizes are totalled to give an overall score and rough language percentages for the entire input document. The chunk boundaries are not refined in any way.

SharpenBoundaries is called only if a vector of per-language text ranges is requested. The assumption is that the user of these text ranges would like fairly careful boundaries between different languages. So the sharpening process scans the SummaryBuffer looking for changes in top language, and then possibly adjusts the starting text offsets and lengths to move the boundaries. We defer the details of this until the later discussion of the intermixed French/English example.

DumpSummaryBuffer This shows the sharpened SummaryBuffer values. They are then totalled across the entire input document to give the DocTote values.

DocTote lists the sums of per-chunk bytes, log probabilities, and reliabilities. In the English example, the line

```
  DocTote::Dump [ 0] en 253B 370p 25300R, 2 chunks scored
```

indicates 253 bytes of English with total probability score of 370 and total reliability score of 25300.

Final result lists the top three languages, their reliability, their percentage of the document text, and the overall summary language.

DumpResultChunkVector shows the optional vector of per-language text ranges. Unlike the previous text shown, this text is the original buffer passed to CLD2, before the preprocessing to remove tags, digits, and punctuation. Each entry in the ResultChunkVector gives original-buffer starting byte offset, byte length, and detected language.

Even more detailed output, Cebuano example

Skipping ahead to the Cebuano detection, we have the first example of mixed language chunks. Three score as Cebuano, but the fourth scores as slightly more likely Tagalog (Currently Cebuano and Tagalog are not treated as close languages, but the probably should be in the future.) This gives us an example of SharpenBoundaries in action.

The text before and after the ceb : tl boundary is shown separated by "^^". Sharpening looks at the probability score differences ceb - tl for each LinearBuffer entry before and after this boundary, expecting mostly positive differences before the best boundary and mostly-negative differences after it. It picks a point of highest contrast, with a strong bias toward word boundaries. In this case, the split is moved from the middle of the word "inusarang" to just before the word "pilipinas". The new split is also shown with "^^" and in addition single carets "^" show where the immediately preceding and following hits are. Finally, the line

= = + _ + - - = # # + = - # # = + = = = # = + ^^ _ _ _ = # _ = = = + = = = _ = + # (scale: #+=-_)

shows the individual hit score differences, ('#' >+2, '+' >0, '=' =0, '-' <0, '_' <-2). In this case, the distinction between Cebuano and Tagalog is pretty vague so many of the probability differences are zero. The more interesting case is the deliberately intermixed French and English, below.

Even more detailed output, French/English example

Near the bottom, our French/English example shows five chunks, two having more French and three having more English. SharpenBoundaries for the first fr : en boundary moves the boundary from the middle of "collections" to just before "this", a perfect result. You can see in the diagram that almost all the entries to the left of the new split favor French and those to the right favor English.

- # # # - = # # # # # # # = # # = - = # # _ # # - - - = = # ^^ _ - - _ _ - _ _ _ + _ _ - = _ _ (scale: #+=-_)

The subsequent en^^fr boundary is moved back from the middle of "européen" to just before "le pays". This is not as good as going left another three words to just before "cet article", but the letter combinations in"article concerne" score as somewhat more likely English than French:

Q:ticl=en.6~fr.4~ca.4 Q:cle_=en.6~fr.4~ca.4 Q:rne_=da.6~no.5~sk.4

Fundamentally, this is the price of not having a large word dictionary for all languages involved.

The final fr^^en boundary is moved from just before "after" to just before "events", an improvement but not perfect. It turns out that none of the letter combinations in "motoring" are scored at all.

Thus ends our quick walkthrough of the debug output available from CLD2. Enjoy.