QDL – A Cue Description Language
Nigel Ward
June 15, 2007
in /n/papers/discover/qdl.tex

1 Background

1.1 Aims

We are interested in discovering response rules such as this (Ward & Tsukahara, 2000):

Upon detection of
  a region of pitch less than the 26th-percentile pitch level and
  continuing for at least 110 milliseconds,
  coming after at least 700 milliseconds of speech,
  providing you have not output back-channel feedback within the preceding 800ms,
  after 700ms wait,
  you should produce back-channel feedback.

Here we have 2 “features” (or “clauses”), each referring to some “properties” of the speech signal. Taken together these constitute a “feature complex” which functions as a “cue”. The entire “rule” expresses a connection between a “cue” and a “response”.

1.2 Current Problem

Our workflow for identifying these, and other prosodic cues to turn-taking (Ward & Al Bayyari, 2006), has been hampered by the weaknesses of our tools. We typically

1. listen to audio and view graphical representation of various properties, with the aim of identifying a feature that recurrently precedes the response of interest.
2. write code in putil.c to detect places where the feature occurs
3. write code in pdisp.c to display these places in didi
4. eyeball the didi display to see whether the feature-detecting code is in fact identifying it as present in places where it really is (and in only those places); if not refine the code
5. incorporate the feature into a predictive rule in putil.c or decide.c
6. eyeball the predictions of this rule in didi; if they are in the wrong places (that is, if they don’t match up with responses), refine the feature
7. write code to expose the key parameters of the feature as command-line arguments to didi and respond
8. systematically vary the parameters to improve performance over the entire corpus, as computed by val

9. deploy it in the realtime version, aizula

10. use it to identify clear examples for use in talks, web pages, or the Back-Channel Trainer

There’s a lot of coding involved: every feature needs code changes to five places: to detect it from the signal, to display it in didi, to incorporate it into a predictive rule, to expose its parameters in didi, and to expose its parameters in respond.

The only truly hard part is the detection code. The rest is mostly done by cut-and-paste, but it does take time and it does make the codebase ugly. It is also common for me to forget to invoke didi (or aizula) with the same parameter values I used for respond, with the result that the display is not showing what I expect, and I sometimes don’t realize this for quite a while.

The solution proposed here is a language for declaratively specifying features, cues, and rules, tentatively to be called QDL, for Cue Description Language.

1.3 Desiderata

QDL must enable the succinct characterization of the prosodic cues relevant to turn-taking and dialog acts. It

• must be human-readable
• must be human-writeable
• must be easy to parse, as it typically will be read out of a file to control the behavior of the programs didi, respond, and aizula. Indeed, these programs should behave, as a default, in the way specified by the file default.qdl in the current directory.
• must be easy to generate, as these description files may be written by a program driving systematic search for the best cue
• must be efficiently executable

2 Inequality-Based QDL

2.1 Examples

See Figures.

For Spanish, the most common cue is characterized by a pitch downslope or low region followed by a pitch rise accompanied by a rate reduction on the last syllable and a drop in energy leading into a slight pause. Quantifying and optimizing this, we obtain best performance with a rule modeling the listener as producing a back-channel 200ms after an utterance by the speaker including
# this is the rule for when back-channels are appropriate in English
# from Ward and Tsukahara 2000

property = pitch-percentile
duration = 110
bound = 26
side = less-than
ripe-after = 700
ripe-until = 700

AND

property = log-energy-1sec-moving-average
duration = 700
bound = is-speaking
side = greater-than
ripe-after = 100
ripe-until = 1500

WITH

minimum-spacing = 700

Figure 1:

\[
\text{if } \text{pitch-percentile} < 28 \text{ over } 110 \text{ paint-from } 700 \text{ paint-to } 700 \text{ and } \\text{if } \text{log-energy-1sec-moving-avg} > \text{is-speaking} \text{ over } 800 \text{ paint-from } 0 \text{ paint-to } 1000 \text{ then } \text{back-channel}
\]

Figure 2: Alternative representation of the English rule. Keywords are in italics.

\[
\text{if } \text{pitch-percentile} < 28 \text{ over } 110 \text{ within } 700 \text{ ago and at least } 700 \text{ ago } \\text{and } \text{if } \text{log-energy-1sec-moving-avg} > \text{is-speaking} \text{ over } 800 \text{ within } 1000 \text{ ago and at least } 0 \text{ ago } \text{then } \text{back-channel}
\]

Figure 3: Alternative representation of the English rule, with the time-ranges expressed relevant to the prediction point. Keywords are in italics.
• a low pitch region lasting for at least 50ms and for no more than 200ms with the pitch continuously below the 28th percentile for that speaker, followed by

• a pitch rise ending above the 75th pitch percentile for that speaker, and lasting at least 80ms and no more than 300ms, and including or followed within 200 ms by

• a lengthened vowel (one lasting at least 100ms), followed within 80 ms by

• a period of silence lasting at least 200 ms.

For Arabic, the feature complex of interest is deemed to be present whenever there is a timepoint which is

C1 part of an utterance which has lasted at least 1.8 seconds,

C2 preceded by a downdash lasting at least 30 milliseconds,

C3 where the pitch in the downdash drops by at least 0.8% every 10 milliseconds,

C4 followed within no more than 600 milliseconds by a pause (low energy region) which lasts at least 150 milliseconds,

C5 not followed by a flat pitch region before the pause, where a flat pitch region is one in which the pitch stays within .4% of the average pitch in that region for a period of at least 80 milliseconds, and

C6 not preceded by another back-channel prediction within 900 milliseconds.

3 Explanation of Fields

property The value is a reference to an internal array in `putil.c`. Currently allowed values of property are “pitch-percentile”, “delta-log-pitch”, “pitch-variation”, “normalized-log-energy”, and “delta-log-energy”. Others may be added in future. In future we may also want to extend the language to allow separate specification of the underlying property and of the scale, as in “underlying = pitch, scale = z-normalized”.

duration in milliseconds. These are always minimum durations; for example, if a region of low pitch lasts for 130 ms, it counts as identifying the feature three times (satisfying the first clause of the English rule three times), at 110ms into the region, at 120ms into the region, and at 130ms in.

bound The meaning will depend on the property, and may be numeric or one of a set of standard threshold values.

side can be “less-than” or “greater-than”. (Currently there seems to be no need for a “within-range” option.)
# this is a simplified (and hopefully equally correct) version
# of the rule for Spanish, from Rivera and Ward 2007

property = pitch-percentile
duration = 50
bound = 28
side = less-than
ripe-after = 600
ripe-until = 1000
AND
property = pitch_percentile
bound = 75
duration = 30
side = greater-than
ripe-after = 500
ripe-until = 900
AND
property = log-energy
duration = 100
side = greater-than
bound = is_vowel
ripe-after = 400
ripe_until = 600
AND
property = log-energy
duration = 200
side = less-than
bound = is-silence
ripe-after = 200
ripe-until = 400
WITH
minimum-spacing = 700
# this is a simplified (and hopefully equally correct) version
# of the rule for Arabic, from Ward and Al Bayyari 2007
# has been speaking for a while
property = log-energy-1sec-moving-average
bound = is-speaking
side = greater-than
duration = 1800
ripe-after = 0
ripe-until = 1500
AND
# downdash
property = delta-log-pitch
bound = -.008 # decrease this much every 10 ms
side = less-than
duration = 30
ripe-after = 900
ripe-until = 900
AND
# a pause
property = log-energy
bound = is-speaking
side = less-than
duration = 150
ripe-after = 0
ripe-until = 1000
AND NOT
# no flat pitch after (or during!) the downdash
property = pitch-variation-over-80ms # computed only if pitch present
bound = .004
side = less
duration = 200
ripe-after = 100
ripe-until = 800
WITH
minimum-spacing = 900

Figure 5:
ripe-after is the point after which a certain response to this cue is appropriate. Similarly ripe-untill is the point up to which a certain response is still appropriate. For contextual components of cues these interval between these may be fairly wide, but for triggering events they will typically be the same, I think, meaning that the time of the response is completely determined. The allowance of a margin of error for matches (between rule predictions and observed responses) is handled not with these but elsewhere, in the parameters to the scoring program \texttt{val}.

(Note that the ripe-after and ripe-untill values can (?) be tweaked to encode facts such as that two clauses of the cue must temporally abut (as in Anais’s rule), and that one clauses must occur before another. This way of expressing such temporal relations is awkward conceptually, but has the advantage of (probably) being easier to program and also easier to tune based on data.

minimum-spacing is not so much about cue detection as such, it is about the response rule, specifically about suppressing too-close responses.

4 Execution Model

Every 10ms do the following.

1. For each clause check if the conditions are true, and if so add 1 to the “clauses-satisfied” array at all future positions from ripe-after until ripe-untill.
   
   Note that this does not require anything like code generation, rather it suffices for each clause just to call some generic cue-detecting function with lots of parameters; or maybe the parameters should be stuffed into some struct and the function should consult that.

2. If the current value of the “clauses satisfied” array is equal to the total number of clauses in the rule, then make a prediction, unless a prediction has already been made within minimum-spacing seconds.

5 Shape-Based QDL

An alternative, is to describe the patterns of interest (features) as shapes. Figure 6 gives an example. The idea here is to measure the extent to which the input, at any given point, matches a specified shape, and if the match exceeds a certain value, we can say that the cue is present.

5.1 Keywords

This can easily be described with similar syntax and some keywords from before, plus some new ones keywords:

\texttt{control-points} determine the shape of the template, by linear interpolation. Each control point has a time (in milliseconds from the template start), a target value (property-dependent), and a leniency.
# a hypothetical component

property = pitch-percentile

control-points = (0 50 40) (20 20 10) (140 20 10) (210 20 20)

threshold = .5

skepticism = .5

ripe-after = 700

ripe-until = 700

Figure 6:

The time of the first control point must be zero. The time of the last control point determines the duration of the template.

The leniency represents how much slack is allowed for matches, at each timepoint. If the leniency is high, this part of the template doesn’t contribute much to the match. If the leniency is low, then a close match is expected, and mismatches count for more (are weighted more heavily). The leniency has the same units as the target value. Conceptually, we’re indicating the allowable margin-of-error at each point, although in fact what really matters is the (weighted) margin of error summed over the entire template.

Leniency is also a way to allow the effective template to be shorter than the nominal one. For example, if the first point of a template has huge leniency, then the signal value at that point effectively contributes nothing to the match score, meaning that the effective template length is shorter.

skepticism is only relevant for pitch-related features: this is the value to use as the instantaneous match in places where the pitch is not defined. Thus, if we’re highly skeptical, we are assuming that points where the pitch detector finds nothing are actually perceived as spoiling the impression; if non-skeptical we assume that such ambiguous regions fit the pattern perceptually.

threshold is the value to use to count this template as matching this part of the signal. This is, in effect, a multiplicative value on all the leniencies. It probably could be omitted.

ripe-before/ripe-after here are measured in milliseconds from the end of the template.

5.2 Execution Model

1. As a pre-processing step, generate a full template by linear interpolation between the control points.

2. Every 10ms we compute the instantaneous match as the absolute value of the distance between the template value and the feature value. Note that we don’t square the distance. If we’re dealing with pitch, at times when the pitch is undefined the distance is given by skepticism.

3. The match distance for the whole template is the weighted average of the instantaneous distances at all its points, where the weights are the reciprocals of the leniencies.
Figure 7: Shape-Template Illustration
4. The match quality is $1 - \text{the match distance}$.

5. The template is considered to match if the match quality exceeds the threshold.

\[
\text{if } \text{threshold} < 1 - \frac{1}{\text{duration}} \sum_{i=0}^{\text{duration}} \frac{|\text{target}_i - \text{actual}_i|}{\text{lenience}_i}
\]

5.3 Discovery

Response-relevant shape-based templates can be discovered, I hope, by fairly simple search. Efficiency is a consideration, so it should probably be done in steps. In each step we use the performance result given by \texttt{val} to select the best template.

1. Start by systematically trying all possible y-values for all the control points. Here we’ll infer the rough shape of the cue.
   
   At this stage hold the leniency and threshold values constant, choosing their values so that the rule makes a prediction only half-dozen times a minute.

   After this stage we do finer local search. The following can be done in any order, and repeated any number of times.

2. finer variation of the y values (for each control point try it a little up and a little down and as is, giving $3**4$ or 81 possibilities)

3. fine variation of the x values (again 81 possibilities)

4. fine variation of the leniencies (here the values are probably independent, so do $3+3+3+3$ possibilities)

5. variation of the threshold

6. variation of ripe-after (generally expecting to narrow down the ripe region)

7. variation of ripe-before (ditto)

6 Relation to the Big Picture

The QDL should be expressive enough but not too expressive.

– If too expressive: it could be messy and slow to find the best rules. It could similarly be hard applying the rules to each dialog: requiring messy programming, and a fair bit of run time. It also makes things less human-readable.

– If insufficiently expressive: it could be inadequate to describe all the patterns and rules of interest. Ultimately it’s a psychological or linguistic question whether a set of inequality-based clauses suffices. Currently this handles everything I know of except the Arabic cadence pattern.

Going from very expressive/generic to very constrained:
1. C
2. Python-controlled C
3. Decision trees (Shriberg & Stolcke, 2004)
4. Olac’s family of filters idea
5. Shape-based templates
6. Thamar’s simple temples (vector-product based, without leniency, x-variation, or ripeness)
7. Inequality-based rules

Expressiveness depends on two things: how much complexity to allow in the clauses, and how much complexity to allow in combining the clauses. Perhaps there’s a trade-off between the two. Compared to decision trees, our clauses are more complex in that they stretch over longer times, but otherwise simpler. The lack of nested clause-combination operators is also definitely simpler.

7 General Assumptions Regarding Learning/Search

I think we need a specific learning method, not a generic one (c.f. decision trees and evolution).

I assume that hill-climbing works for all parameters (?)

I assume that if A AND B is a valid cue, then either A or B will have some validity by itself, so that an initially useful feature can always be found.

I assume that the greedy approach will work, so that we can build up a rule clause-by-clause (feature-by-feature).

We may need two learning models, one for discovering the minimal set of features for a cue, and one for determining all the features. The distinction arises because some features are largely redundant. For example, if the pitch stays low for a time then the pitch variation is guaranteed to be low too; and if there is a sharp pitch drop there will also be a sharp energy drop. For synthesis we need the maximal description, but for recognition and teaching, the minimal one should suffice. Note that for redundant cues, we probably should prefer the pitch-based ones, since they are perceptually salient and relative robust to compute.

References

