BASIC USAGE

INTRODUCTION

Basically you *could* use the engine's REST API in the simplest case if you have any application that inputs some data, and based upon the input then outputs a choice or judgment as an English sentence. But naturally just because you *could* use the API doesn't mean you *should.*

So what is a judgment (herein) and why should you use the API?

A judgment is a decision, it is output, it is the results returned by your software given its input. So in the first place a judgment can simply be thought of as "a sentence that your app generates based on the input." However, in a true judgment a decision made could realistically be other than what it is. It could be a variation within certain sometimes ill-defined boundaries. A true judgment is something *you* make and it may often be based upon a certain *feel* you have which might be difficult or impossible to encode. A basic feature of the server is to be able to capture *your* style of feel for judgments and then be able to put that to work on new cases in software.

Because a judgment often contains a clearly defined component along with a gut feel component -- in other words, the judgment is both objective and subjective -- the API is set up to handle a workflow in which objective and subjective are accommodated.

The API organizes by case. A case has one or more judgments by subject along with an optional final judgment about the entire case.

Perhaps your app just makes one decision per case. In more complicated scenarios your app makes multiple decisions about different subjects – that is, inputs more data and makes choices about other relevant subjects – and then reaches a final judgment based upon the preceding judgments about different subjects.

Using the API would be a good choice if:

- Your app's output judgments tend to have *lots of combinations*,
- There is probably an objective component and a subjective component to judgments,
- There is typically a clear sequence of judgments based upon relatively independent subjects leading to a final summary judgment,
- Your app could realistically compute more than one valid judgment *given the same input*, and you would like to test alternatives.
- It is not practical to completely hardcode for all possibilities,
- You want to use the engine rather than create a series of your own ML models and code.

JUDGMENT STRUCTURE

The overall structure is by cases. A case can contain multiple judgments by subject leading to a final summary judgment about the case. It is hoped that this generic structure will cover a lot of cases.

Judgments can be submitted to the server using three levels.

The first level consists of one or more simple judgments about a subject in your application domain along with your application input which led to the judgment. The second level is a judgment about the same subject taking into consideration each immediately preceding level one judgment. The third level is a judgment based upon all prior judgments in the case (ie., all prior level one and level two judgments). The levels are called level 0, 1, and 2 in the system (it starts on zero).

Judgments can be classified and submitted as either category ‘system 1’ or ‘system 2’. The system 1 type is taken by the server to be true and realistic. The system 2 type requires deliberation by the server. Most of the initial training is done by submitting system 1 type records.

The overall scheme of the server: it learns the patterns in your system 1 cases so it can apply the knowledge gained to different variations of new system 2 cases.

Level 0 judgments are always system 1 (taken as true by the server). Level 1 and level 2 judgments can be either system 1 or system 2.

WORKFLOW

A single algorithm can be used for training and querying as follows. This could be coded using a low-code facility for speed, perhaps, or any sort of program.

For readability please page down…

Submit a transaction to the server to create a new case

For each relevant subject within the case:
Create level 0, system 1 judgment(s) and post to the server
Your app generates none, some, or all of the level 1 judgment about the subject
If the user just proceeds to next screen accepting your generated judgment,
Post a level 1, system 1 judgment
Else repeat until user satisfied:
User can alter the judgment by choosing one of three methods:
Insert a '?' alone into the level 1 judgment, post system 2 to trigger generation
Insert '…' right after a partial level 1 judgment, post system 2 to trigger judgment completion
Or modify in any way or leave the judgment the same, post level 1, system 2
The user can now submit a query providing the case number, the server returns either "realistic" or "unrealistic" regarding the judgment which may be modified (eg., a "?" or "…" will trigger a generated judgment)
User can choose:
If satisfied, continue
else if unsatisfied retract the level 1 judgment

If a final judgment for the entire case is desired:
Your app generates none, some, or all of the level 2 judgment about all case subjects
If the user just proceeds to next screen accepting your generated judgment,
Post a level 2, system 1 judgment
Else repeat until user satisfied:
User can alter the judgment by choosing one of three methods:
Insert a '?' alone into the level 2 judgment, post system 2 to trigger generation
Insert '…' right after a partial level 2 judgment, post system 2 to trigger judgment completion
Or modify in any way or leave the judgment the same, post level 2, system 2
The user can now submit a query providing the case number, the server returns either "realistic" or "unrealistic" regarding the entire case, and the final judgment which may be modified (eg., a "?" or "…" will trigger a generated judgment)
User can choose:
If satisfied, continue
else if unsatisfied retract the level 2 judgment
When satisfied user can mark entire case for training.

Please see the API documentation for specifics.