Dustin Jones

Ashok Goel

CS-7637

2016-12-08

Final Exam

In the day to day world of human interaction, people are constantly learning new information. While some of that information is simply taken at face value, a significant portion of that information is vetted before being accepted as fact. One common way of vetting that information is by seeking explanation from the source.

Design of an RPM Agent

For Project 3 I designed an agent modeled after the rules given for solving Ravens Progressive Matrices (RPMs). The rules define that there exist relationships both horizontal and vertical across rows and columns respectively, and that they need not be the same. By using these rules, the agent breaks a problem down into sets of 3 figures (by row and by column) into entities known as `FigureSeries`. To help identify patterns across `FigureSeries`, figures are broken down into sets of disconnected `Pixels` (simple objects containing an 'x' coordinate and a 'y' coordinate), which can be compared both within a figure and across figures. The agent uses basic Dark Pixel Ratio (DPR) and Intersection Pixel Ratio (IPR) metrics, as well as simple attributes such as number of objects in a figure, and objects being 'solid', 'hollow', or 'surrounding' (in the case of a pair of objects) to compare the similarity of entire figures and disconnected `Pixel` sets.

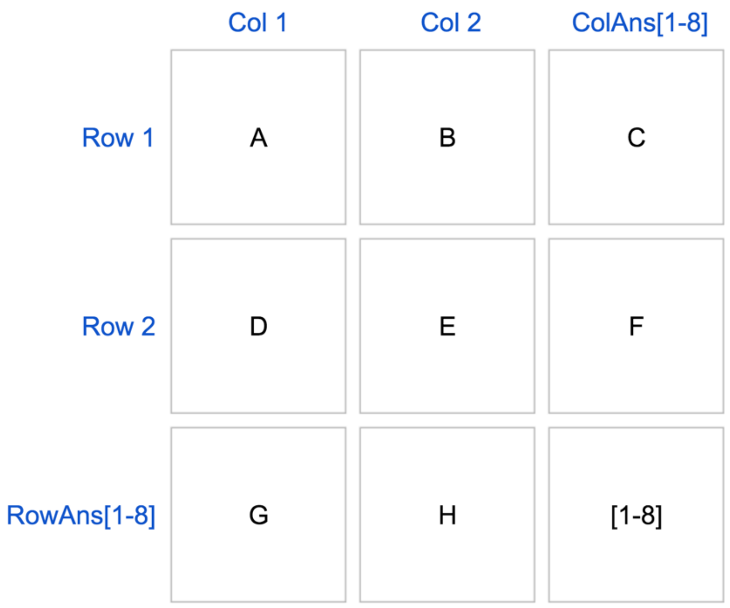

Fig. 1 Assignment of FigureSeries (Jones)

At the core of its functionality, the agent executes a suite of approximately 20 tests, most run twice, once for horizontal relationships, once for vertical relationships. These tests are run for every answer, each test returning the set of answers that exactly satisfy its requirements without 'guessing', and able to return up to and including all 8 potential answer figures. Each test votes for answers, with votes for 0 figures or votes for all 8 figures not counting towards (or against) the overall tally. After every test is executed, the results are tabulated and if a single answer holds the simple majority of votes, it is selected by the agent. If multiple answers are tied for the majority, and the number of tied answers is below a certain risk threshold (meaning 4 or less answers out of 8 possible answers tied) then one of the answers is selected at random. However, if either no answers are voted for, or if the number of tied answers is above the risk threshold, the agent simply skips the problem.

Adaptation into an Explainable Agent

Because the agent is essentially comprised of a static list of tests, one of the simplest ways to build trust is to create a user manual. By creating a user manual with a high level overview of the agent's operation and providing a comprehensive list of every test paired with visual examples (and potentially walkthroughs of their behavior) for each, the agent has a starting point from which to explain its reasoning. The agent will also include a simplistic natural language interface that will allow end-users to interact with the agent and make inquiries about its reasoning processes.

In much the same way as cars and trucks operate, the same design pattern can be applied to the agent. For example, there are several gauges and indicators available to drivers to let them know at a high level the status of their vehicle. All the indicators are documented to varying degrees inside the user manual, some are very basic in their functionality such as the unbuckled seatbelt indicator, others are slightly more detailed such as the fuel gauge, and others still have their information abstracted from the driver such as the check engine light. By connecting directly to the computer of a vehicle via an ODBII connector, it is possible to get even more detailed diagnostic information in the way of error codes and precise instrument measurements.

Building upon the automotive explanation model analogy, the agent will capture and record metadata during the execution of known (Basic and Challenge) problems to allow for explanation at both a simplistic, and at a highly detailed level depending on the use-case. The agent will capture data beginning at the problem level, progressing down through details about individual tests, and details about individual figures. Storing each of these sets of metadata in frames will allow the agent to make further inferences by correlating multiple sets of metadata within the context of a single problem and across multiple problems.

Framing Problem Metadata

The frames specifically related to Problems associated with the metacognitive layer of the agent will be constructed as shown in Fig. 2. There will be frames representative of problems, tests, figures, and objects.

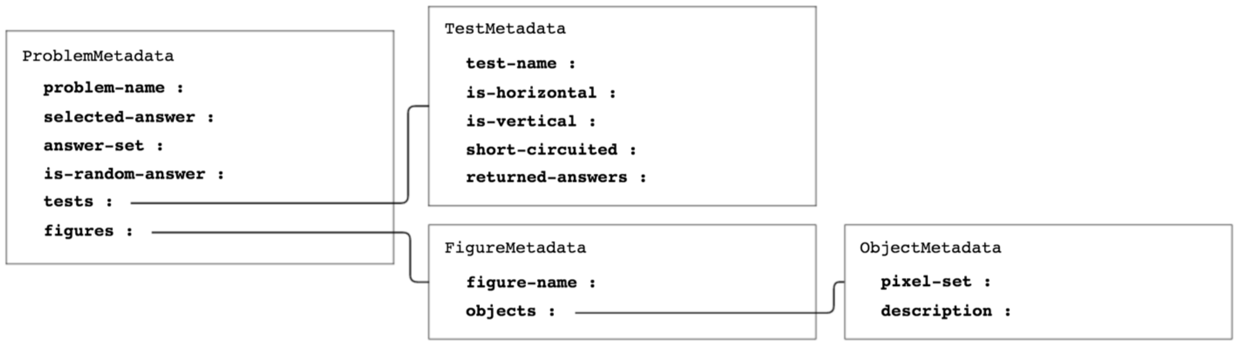

Fig. 2 Metadata frames

The metadata associated with `ProblemMetadata` frames will be representative of the following questions:

- What is the name of the problem?
- What answer did the agent select?
- Was that answer selected randomly from a set?
- How many answers were in that set?

The metadata associated with `TestMetadata` frames will be representative of the following questions:

- What is the name of the test?
- Is it testing the horizontal (row `FigureSeries`) data for this problem?
- Is it testing the vertical (column `FigureSeries`) data for this problem?
- Did the known (non-answer row/columns) data fit the pattern of this test?
- What answers did the test identify as matching its pattern?

The metadata associated with the `FigureMetadata` and `ObjectMetadata` frames will be representative of the following questions:

- How many (disconnected) objects are present in this figure?
- What do the objects in this figure look like?

While each of these pieces of metadata are somewhat useful individually, their true power comes from being able to aggregate their values.

Framing User Interaction

As the focus of the agent is not primarily on interacting with the user in an adaptive manor, the natural language processing component will be simple and limited, framed as shown in Fig 3. A user will pose questions to the agent, first specifying the name of the problem in question which will be persisted until a new problem name is specified. Using basic keyword matching (with support for synonymous keyword identification), the agent will identify the keywords within a `UserInquiry`. A predefined dictionary of `Questions` will be associated with certain keywords and will be mapped to metadata regarding the problem, as well as the tests and answer figures for that problem.

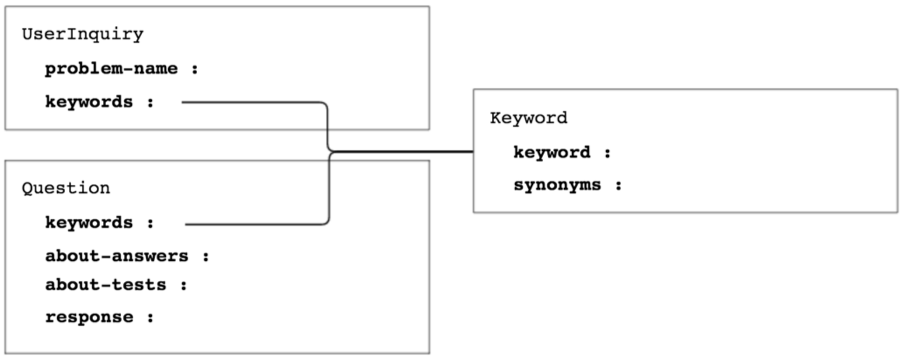

Fig. 3 User interaction framing

Answering Questions

As several sample questions have been provided, the agent will focus on answering them to the best of its ability.

(1) What is the object in the eight images displayed in the matrix?

Because the agent relies upon the rules of RPMs - pattern identification across rows and across columns - it has no use for shape identification across an entire matrix. It also does not attempt to assign commonly known names to objects (e.g. square, triangle, circle). Given these conditions about the agent, it would be unable to answer the question as stated, however it would be possible for it to answer a question like "What objects are displayed in figure A of the matrix?". To answer this question, the agent could provide the answer in 2 ways: verbally and visually. To provide the answer visually, the agent would simply query the `FigureMetadata`, returning and rendering the `pixel-set` data from all associated `ObjectMetadata` frames. To provide the answer verbally, the agent would query the `description` field from the previously mentioned metadata frames, this description having been populated by the few characteristics that various tests use in their computations (namely 'hollow', 'solid', 'surrounding'). In general, the answers returned by this particular line of questioning would be largely unhelpful to an end-user due to the overall problem solving methodology of the agent.

(2) What is the relationship between the three images in the top row of the matrix?

(3) What is the relationship between the three images in the middle column?

Both questions 2 and 3 are effectively the same with respect to the agent, and could be answered by the same `Question`, however not in their current form. Every test reasons over the entire set of `FigureSeries` per direction (horizontal or vertical), and doesn't differentiate between which set (first/top/left, middle) is being analyzed. The tests do however allow for a particular pattern to 'short circuit', whereby if the pattern does not fit the known rows (top and middle) or columns (left and middle) then it will skip the process for testing answers.

To answer these questions, the more appropriate phrasing for the questions would be "What is the horizontal relationship between the known rows of the matrix?" and "What is the vertical relationship between the known columns of the matrix?". The response to both questions would be the set of tests that did not short circuit for the direction of the test.

To answer question 2, the agent would query the `TestMetadata` for the following values:

```
    is-horizontal : true
    short-circuited : false
```

To answer question 3, the agent would query the `TestMetadata` for the following values:

```
    is-vertical : true
    short-circuited : false
```

Filling the `test-name` into the `response` field of the `Question` will result in a response like "Relationship(s) between the known rows have been identified by the following tests: [TestPattern1, TestPattern3, …TestPatternN]".

(4) What is the difference between the #3 and #4 as potential answers?

There are 2 possible ways of viewing this question. The most simplistic answer would be to analyze the number of votes for each answer and present them to the user. To answer in this manor, the agent would query the `ProblemMetadata` for the following values:

```
number of votes for 3 = answer-set[3]
number of votes for 4 = answer-set[4]
```

Filling these values into the `response` field of the `Question` would result in a response like "Answer 3 received 1 vote, and answer 4 received 2 votes".

If on the other hand the user was seeking a more detailed response, the agent would return the symmetric difference of tests (the tests that only vote for one answer or the other, but not both) by querying the `TestMetadata` for the following values:

```
returned-answers contains 3 and not 4
returned-answers contains 4 and not 3
```

Filling the `test-name` into the `response` field of the `Question` will result in a response like "The following tests only voted for answer 3: [TestPattern1, TestPattern2], while the following tests only voted for answer 4: [TestPattern3]".

(5) Why did you select #5 as the answer?

Like the previous questions, there are 2 possible ways of viewing this question. The most simplistic answer would be to analyze the number of votes, how it was voted for and present them to the user. To answer in this manor, the agent would query the `ProblemMetadata` for the following values:

```
number of votes for 5 = answer-set[5]

is-random-answer
```

Filling these values in the `response` field of the `Question` will result in a response like: “Answer 5 received the majority of votes at 6 votes and was not selected randomly”. If on the other hand the user was seeking a more detailed response, the agent would return the list of tests that voted for that particular answer by querying the `TestMetadata` for the following values:

```
returned-answers contains 5

is-random-answer
```

Filling the `test-name` into the `response` field of the `Question` will result in a response like “The following tests voted for answer 5: [TestPattern1, TestPattern2, TestPattern6, TestPattern7, TestPattern10]”.

(6) Why is #3 not the correct answer?

Like the previous questions, there are 2 possible ways of viewing this question. The most simplistic answer would be to analyze the number of votes, how it was voted for and present them to the user. To answer in this manor, the agent would query the `ProblemMetadata` for the following values:

```
number of votes for 3 = answer-set[3]

number of votes for selected-answer =

     answer-set[selected-answer]

is-random-answer
```

If the answer was not selected randomly, filling in the `response` field of the `Question` will result in a response like: "Answer 3 did not receive the majority of votes with only 3 votes counted".

If the answer was selected randomly, and the number of votes for answer 3 was the same as the number of votes for the selected answer, the filling in the `response` field of the `Question` will result in a response like: "Answer 3 was tied for the majority of votes with 3 counted, but another answer was chosen randomly".

Works Cited

Joyner, David A., Darren Bedwell, Chris Graham, Warren Lemmon, Oscar Martinez, and Ashok K. Goel. "Using Human Computation to Acquire Novel Methods for Addressing Visual Analogy Problems on Intelligence Tests." Proceedings of the Sixth International Conference on Computational Creativity. Provo, Utah. 2015. Web.

Jones, Dustin. "Project 3 Reflection," submitted, 2016.