Divisi For Dummies: Overview

Fry, Dec 19, 2012

Divisi is a software package for extracting knowledge from Conceptnet. It helps a program reason over the common sense knowledge in a conceptnet. Divisi is implemented as a library written in Python and called using Python. Although it is powerful, it’s not so easy to use. Divisi For Dummies (d4d) is a layer on top of Divisi to make it easier for programmers to work with conceptnet. It eliminates the need for programmers to understand the matrices used to infer common sense. D4d initializes these matrices behind the scenes and keeps them there. D4d instances initialize more data structures than you may end up needing, but then you don’t have to worry about what you do need.

D4d simplifies three areas of working with Conceptnet. The first is in taking text from a variety of sources and ‘normalizing” it so that it can be used by Divisi. This normalizing happens on a number of levels. Relations are normalized to one of Conceptnet4’s

24 relations by accepting over 100 different “spellings” of those relations as well as accepting any letter case. Concepts are normalized to known concepts in conceptnet4 if you use the ‘normalize\_concepts=True’ parameter to many methods. Assertions can be built up from arrays of concept-relation-concept or from sentences, ie “A pet is an animal.” Assertion sets can be normalized from arrays of assertions or paragraphs of sentences, with flexibility about how those assertions are separated from each other. D4d can also read csv files exported from spread sheets with a flexible configuration for the format of data in those spreadsheets. D4d can also accept input from the “Common Concensus” game in its JSON-formatted data output. D4d also simplifies the “blending” of two conceptnet knowledge bases, especially when one of them is the large “c4” knowledge base. D4d stops well short of actual understanding of content when transforming it into properly formatted assertions, but it removes a lot of the nit-picking kind of transformations you have to understand when using straight Divisi.

The second area of simplification is in querying a conceptnet knowledge base. Here’s an example of d4d code: d4d.c4.how\_true\_is(“A dog is a pet.”), which returns a “score” as a confidence number. Sometimes it is more useful for a program to separate out the components of an assertion and make a call like

d4d.c4.how\_true\_is(“dog”, “isa”, “pet.”) D4d allows that too. Other methods include “how\_related\_are”, “how\_similar\_are”, “similar\_concepts\_to”, “related\_concepts\_in\_category” and a few more.

Here’s an example lifted from the “books” file included in the d4d release:

d4d.normalize("books", """

Knowledge is part of books.

A Page is part of a book.

Books are used for reading.

Knowledge is part of a magazine.

A page is part of a magazine.

Magazines are used for reading.

Ice has the property of cold.

Page is part of a newspaper.

Newspapers are used for reading.

""", separator=”.”, principle\_component\_count=3)

d4d.books.how\_true\_is("knowledge", "PartOf", "newspaper")

First notice a couple of things about the knowledge base.

1. “Ice has the property of cold.” isn’t related to the rest of the assertions, but that’s ok, it doesn’t screw them up either.
2. The assertion that “Knowledge is part of a newspaper.” is NOT in the knowledge base.

None the less, the score for our how\_true\_is call is 0.48.

This is not as high as if we had queried for a direct assertion but well above zero. How\_true\_is uses the magic of AnalogySpace to infer that. How? Because it can infer that newpapers are similar to books, and since knowledge is part of books, then knowledge is probably also part of newspapers. This is just the kind of common sense reasoning that humans are intuitively good at but computers, up until now, have been terrible at. Despite our human skill, most of us don’t know HOW we perform this clever inference. This lack of meta-knowledge about how we do things is why it is so hard to teach computers to do those same things.

Rather than leave programmers with just such high-level tools, d4d also provides a number of utilities that help manipulate real-world knowledge. This gives programmers the flexibility to write algorithms we’ve yet to think of, which is the mark of any truly powerful library.

The third area of simplification involves “visualization” of conceptnets. A set of functions do what we call “abnormalization” or transforming formal assertion vectors into prose. We can also show a conceptnet as a multi-dimensional interactive space to give programmers an overall feel for what’s in a knowledgebase. We have plans to extend this area of d4d in the near future because visualization is so useful for demos, debugging and sometimes actual output for end users.

One area we like to apply common sense reasoning to is in facilitating the user interface for end user applications. Such programs won’t get built if the programmer interface to common sense reasoning itself isn’t easy. That’s why we’re building d4d!