What is an AI assistant?

Tomas Petricek

July 26, 2018

1 Introduction

The idea of an *AI assistant* has been discussed in many contexts in the AIDA project. The Wrattler notebook system aims to provide a platform through which AI assistants can be easily used; the datadiff project developed an AI assistant which automates the process of reconciling inconsistencies between pairs of tabular datasets. However, we never clearly defined what an AI assistant is. This document aims to collect some notes to help answer this question. In the current version, these are mostly based on the needs of Wrattler and datadiff integration in Wrattler – as such, it clearly does not give the final answer, but merely one (limited, but hopefully useful) perspective.

A simple way of trying to address the data wrangling problem would be to build a set of functions that take some dirty datasets and produce one or more clean datasets. The user could then call these functions on their dirty data and would get a clean result (conceivably, this could also be further automated). However, this simple approach has two problems. First, real-world data sets often contain numerous corner cases that require some human insight. Second, we want the data analysis to be transparent and be able to understand how have the data wrangling problems been solved. A simple function does not address these because it does not encourage human interaction[1] and it is not transparent – it produces a result without indicating how and why.

2 What makes AI assistants unique

AI assistants address the above issues as follows. First, AI assistants are interactive. Given some input data, they perform some analysis and provide the user with several options to choose from (those can be most likely solutions to some problem, possible ways of resolving ambiguous corner cases, or a choice of next steps the AI assistant can follow). The options can be a finite list (where the user has to choose one) or open-ended (where the user can specify some input). They should also be sorted by likelihood (if we always choose the most likely option, the AI assistant will behave as a fully automatic function).

Second, AI assistants do not directly transform data, but instead, generate scripts (or, more formally, expressions) in simple languages (domain-specific languages, or DSLs)

[1] At best, users can run the function again with different parameters, but they get no suggestion about suitable values of the parameters.

that define how data should be transformed. The AI assistant can still use sophisticated machine learning algorithms (that do not produce an explanation of how they work) behind the scenes, but the end result will always be a simple script that can be reviewed and understood by the user.

Note that each AI assistant can use its own domain-specific language – the language should be as small as possible, i.e. it should be able to express the data transformations that the AI assistant can recommend, but nothing more.

The combination of interactivity and domain-specific languages allows us to do a number of interesting things:

- Scripts generated by AI assistant can be translated to R or Python (and it is easy to add support for other languages), which means that one can use an AI assistant during data analysis and exploration, but then end up with code that can be used in production in any programming language.
- The interactive nature of AI assistants makes it possible to give immediate feedback to users and let them correct potential issues early. For example, in Wrattler, we display the options that AI assistant provides together with a preview of results (of applying the generated expression to a sample dataset).

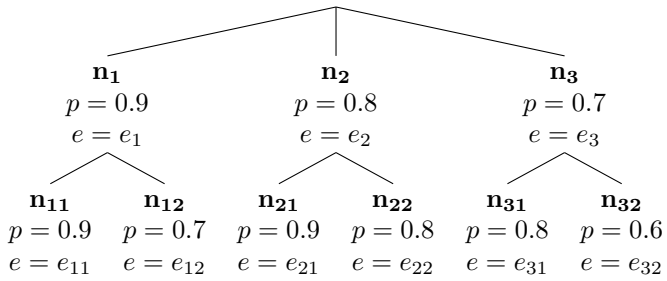
3 AI assistants as trees

We can think of an AI assistant as a function that takes some input data and generates a tree where each node offers a finite list of options that the user can choose from. (For simplicity, we ignore the possibility of open-ended options where the user specifies some input and we also ignore the fact that the AI assistant might take some initial parameters in addition to the input). Formally:

$$
\begin{array}{rcll}
a & = & \mathsf{Data} \to t & \text{(AI assistant)} \\
t & = & (n_1, p_1, t_1), \ldots (n_k, p_k, e_k, t_k) & \text{(Provided options)} \\
eval & = & \mathsf{Data} \times e \to \mathsf{Data} & \text{(Evaluator for the DSL)}
\end{array}
$$

An AI assistant a is a function that takes some input data and returns a root tree node t. A tree node contains a list of options (n_i, p_i, e_i, t_i) where n_i is a name of the node (shown to the user, but otherwise unimportant), p_i is a number that gives the likelihood of the option (between 0 and 1), e_i is an expression in the domain-specific language that represents the data transformation attached to the node and t_i is a sub-tree that can be empty or can offer more follow-up options. Note that this sub-tree can be generated lazily – it is only needed if the user chooses the option with which it is associated and so we do not need to evaluate the full tree upfront.

The nature of expressions e depends on the particular AI assistant and so we leave that abstract for now. However, we certainly need to be able to apply the transformation on sample dataset, which is captured by the $eval$ function, which takes data together with a DSL expression and returns transformed data. An (abstract) example tree might look as follows:



Here, the root offers three options represented by nodes n_1, n_2, n_3. The most likely one as recommended by the AI assistant is n_1. The expression attached to the node is e_1 and when the user selects the node, we can display a preview using sample data $sample$ by evaluating $eval(sample, e_1)$.

Assuming the user chooses the most likely option n_1, they will then be presented with two subsequent options, n_{11}, n_{12}. Even though the AI assistant marks the first node as more likely, the user may review the previews $eval(sample, e_{11})$, $eval(sample, e_{12})$ and decide that the latter is actually the correct data cleaning operation. They can then choose the node n_{12} as the final one use the cleaning script represented by the DSL expression e_{12}.

4 Examples of AI assistants

What would a sample AI assistant look like when seen through the above model? We consider two examples. The first is hypothetical AI assistant for extracting data from CSV files. The second is the datadiff assistant. For datadiff, we discuss the current implementation in Wrattler, but also briefly sketch a possible improvement (that improves the interactivity aspect).

4.1 Cleaning CSV files

As an example, consider the following broken CSV file. The fle starts with two metadata lines (one specifying the name and another specifying copyright information). This is followed by headers and actual data.

```
Counts of things
Copyright (c) Poor data export company, London, UK
Name, City, Count
Joe, London, 3
Jane, Edinburgh, 16
Jim, Cardiff, na
```

A hypothetical minimalistic AI assistant for cleaning CSV files could generate tree with two levels of nodes. In the first level, it will aim to infer where the actual data is in the CSV file (and it will offer its best guesses to the user). In the second level, it will aim to infer the types of the columns.

The domain-specific language of data cleaning operations consists of two functions named $range$ and $types$ that specify the range of data in the file and the types of the columns: $range(lines = l_n \ldots l_m, cols = k)$ denotes that the data starts on line l_n and ends on line l_m and there are k columns; $types(t_1, \ldots, t_k)$ specifies that the types of the k columns are $t_1, \ldots, t_k$. The types are either string or int.

Given the above file as an input, the AI assistant might produce the following tree:

$$
\begin{aligned}
&p = 0.9,\ e = range(lines = 2 \ldots 6, cols = 3) \\
&\quad \longrightarrow p = 0.9,\ e = range(lines = 2 \ldots 6, cols = 3); types(\mathsf{string}, \mathsf{string}, \mathsf{string}) \\
&\quad \longrightarrow p = 0.3,\ e = range(lines = 2 \ldots 6, cols = 3); types(\mathsf{string}, \mathsf{string}, \mathsf{int}) \\
&p = 0.7,\ e = range(lines = 3 \ldots 6, cols = 3) \\
&\quad \longrightarrow p = 0.8,\ e = range(lines = 3 \ldots 6, cols = 3); types(\mathsf{string}, \mathsf{string}, \mathsf{int}) \\
&\quad \longrightarrow p = 0.7,\ e = range(lines = 3 \ldots 6, cols = 3); types(\mathsf{string}, \mathsf{string}, \mathsf{string}) \\
&p = 0.3,\ e = range(lines = 1 \ldots 6, cols = 1) \\
&\quad \longrightarrow p = 0.9,\ e = range(lines = 1 \ldots 6, cols = 1); types(\mathsf{string})
\end{aligned}
$$

At the first level, the AI assistant gives us three options. The first, most likely, one skips just the first line and identifies that there are three columns (perhaps because the second line contains two commas and can thus be interpreted as data row). The second skips first two lines and recognises three columns. The AI assistant might find this less likely, but it is the option we want to choose as users. Finally, the third option takes all lines, but then it has to join all columns into one which is unlikely.

The user can choose the second option, at which point a user interface can evaluate the expression $range(lines = 3 \ldots 6, cols = 3)$ and show a preview of the cleaning result. Next, the user has a choice of two ways of inferring types – the most likely one (and the right one) is that the third column has a type int (which means that the value na in the last row will be treated as missing).

The resulting expression $range(lines = 3 \ldots 6, cols = 3); types(\mathsf{string}, \mathsf{string}, \mathsf{int})$ can then be applied to the input data using the $eval$ function, which gives us columns Name of type string, City of type string and Count of type int with data:

$$
\begin{aligned}
&(\text{"Joe"}, \text{"London"}, 3) \\
&(\text{"Jane"}, \text{"Edinburgh"}, 16) \\
&(\text{"Jim"}, \text{"Cardiff"}, NaN)
\end{aligned}
$$

4.2 Datadiff (now)

4.3 Datadiff (better)

5 Evaluating AI assistants

5.1 Overall

Following the best probability, we get the best result (this is equivalent to running a function)

5.2 Interactivity

Imagine there are some annomalies that we cannot detect. Does the AI assistant provide an easy way for the user to address them?