On Discovering Data Preparation Modules Using Examples

Khalid Belhajjame^(✉)

PSL, Université Paris-Dauphine, LAMSADE, Paris, France
`khalid.belhajjame@dauphine.fr`

Abstract. A major issue that arises when designing data-analysis pipelines is that of identifying the services (or what we refer to as modules in this paper) that are suitable for performing data preparation steps, which represents 80% of the modules that compose data analysis workflows. Such modules are ubiquitous and are used to perform, amongst other things, operations such as record retrieval, format transformation, data combination to name a few. To assist scientists in the task of discovering suitable modules, we examine, in this paper, a solution that utilizes semantic annotations describing the inputs and outputs of modules together with data examples that characterize modules' behavior as ingredients for the discovery of data preparation modules. The discovery strategy that we devised is iterative in that it allows scientists to explore existing modules by providing feedback on data examples.

1 Introduction

Despite the impressive body of work in data management on data preparation tasks, it is recognized that there is not a single generic one-shop-stop solution that can be utilized by the scientists to prepare their data prior their analysis. Instead, data preparation tasks are numerous, can be difficult to generalize (e.g., data cleansing, data integration), and tends to vary depending on the processing tasks at hand, but also on the semantic domains and the format of the data subject to processing. As a result, scientists tend to develop their own program/script using their favorite language, e.g., Python, R or Perl, to prepare their data. This operation is time-consuming and recurrent since sometimes the scientist has to redevelop data preparation scripts that s/he has previously performed on the same or similar data.

To overcome the above problem, a number of researchers have been calling for the creation of repositories dedicated to data preparation modules with the view to save the time scientists spend on data preparation to allow them to focus their effort on the analysis tasks. Examples of such repositories are BigGorilla¹, an ecosystem for data preparation and integration, Bio.Tools², a catalogue

¹ <https://www.biggorilla.org>.

² <https://bio.tools>.

which provides access to, amongst other things, services for the preparation of bioinformatics data, and Galaxy tools³.

In this paper, we set out to examine the problem of querying data preparation modules. Specifically, the objective is to locate a module that can be perform a data preparation task at hand, if such a module exists. Semantic annotations can be used to reach this objective [9]. A module is semantically annotated by associating it to concepts from ontologies. Different facets of the module can be described using semantic annotations, e.g., input and output parameters, task and quality of service (QoS). In practice, however, we observe that most of semantic annotations that are available are confined to the description of the domain of input and output parameters of modules. Annotations specifying the behavior of the module, as to the task it performs, are rarely specified. Indeed, the number of modules that are semantically described with concepts that describe the behavior of the module lags well behind the number of modules that are semantically annotated in terms of the domains of the input and output parameters, e.g., in BioTools. Even when they are available, annotations that describe the behavior of the module tend to give a general idea of the task that the module implements, and fall short in describing the specifics of its behavior. For example, the modules in BioTools, which is a registry that provides information about data preparation modules, are described using terms such as *merging* and *retrieving*. While such terms provide a *rough* idea of what a module does, they do not provide the user with sufficient information to determine if a it is suitable for the data preparation at hand. The failure in crisply describing the behavior of scientific modules should not be attributed to the designers of task ontologies. Indeed, designing an ontology that captures precisely the behavior of modules, without increasing the difficulty that the human annotators who use such ontologies may face thereby compromising the usability of the ontology, is challenging.

To overcome this issue, we examine in this paper a solution that utilizes semantic annotations describing the inputs and outputs of modules together with data examples that characterize modules' behavior as ingredients for the discovery of data preparation modules. Given a module m , a data example provides concrete values of inputs that are consumed by m as well as the corresponding output values that are delivered as a result. Data examples are constructed by harvesting the retrospective provenance of modules' executions. They provide an intuitive means for users to understand the module behavior: the user does not need to examine the source code of the module, which is often not available, or the semantic annotations, which require the user to be familiar with the domain ontology used for annotation. Moreover, they are amenable to describing the behavior of a module in a precise, yet concise, manner. It has been shown in [2] that data examples are an effective means for characterizing and understanding the behavior of modules. We show in this paper that data examples can also be used to effectively and efficiently discover modules that are able to perform a data preparation task of interest.

³ <https://galaxyproject.org/tools>.

GetRecord.i	GetRecord.o
P17110	<pre>>sp P17110 CH36_CERCA Chorion protein S36 OS=Ceratitis capitata MNCFLFTLFFVAAPLATASYGSSSGGGSSSYLSSASSNGLDELVQAAAGGAQQAGGTI TPANAEIPVSPAELVARLNQVQAQLQALNSNPVYRNKNSDAIESLAESSLASKIRQGN NIVAPNVIDQGVYRSLVPSGQNNHQVIATQPLPPIIVNQPALPPTQIGGGPAAVVKAA VIYKIKPSVIYQQEVINKVPTPLSLNPVYKVKPGKKIDAPLVPGVQQNYQAPSYGSS YSAPAASYEPAPAPSYSAAPAQSYNAAPAPPSYSAAPASASYGAAPSASYDAAPAS SYGSPQSSSYGSAPPASGY</pre>

Fig. 1. Data example.

It is worth noting that a number of systems have been developed recently to facilitate data preparation tasks, including Trifacta⁴, NADEEF [3], Tamer [8] and VADA [6]. These systems come with a number of functionalities that covers, amongst other things, format transformation, data deduplication and data repair. They are primarily targeted for end-users (be they domain expert or not), who would like to use a GUI to clean a single tabular dataset (mainly in relational form or CSV). In our work, we target scientists who wish to *programatically* process one or multiple datasets, in any format (relational, CSV, text, JSON, etc).

The paper is structured as follows. We start by introducing background information regarding data examples and how they are generated for characterizing modules based on retrospective provenance of modules’ executions (in Sect. 2). We go on to present our solution for module discovery (in Sect. 3), and close the paper (in Sect. 4).

2 Background

For the purposes of this paper, we define a data-preparation module by the pair: $m = \langle \text{id}, \text{name} \rangle$, where **id** is the module identifier and **name** its name. A module m is associated with two ordered sets $\text{inputs}(m)$ and $\text{outputs}(m)$, representing its input and output parameters, respectively. A parameter p of a module m is characterized by a structural type, $\text{str}(i)$, and a semantic type, $\text{sem}(i)$. The former specifies the structural data type of the parameter, e.g., **String** or **Integer**, whereas the latter specifies the semantic domain of the parameter using a concept, e.g., **Protein**, that belongs to a domain ontology [5].

A data example δ that is used to describe the behavior a module m can be defined by a pair: $\delta = \langle I, O \rangle$, where: $I = \{\langle i, \text{ins}_i \rangle\}$ and $O = \{\langle o, \text{ins}_o \rangle\}$. i (resp. o) is an input (resp. output) parameter of m , and ins_i and ins_o are parameter values. δ specifies that the invocation of the module m using the instances in I to feed its input parameters, produces the output values in O . We use in what follows $\Delta(m)$ to denote the set of data examples that are used to describe the behavior of a module m .

Example 1. To illustrate how data examples can be used to understand a module behavior, consider the module **GetRecord**, which has one input and one output. Figure 1 illustrates an input instance that is consumed by **GetRecord** and

⁴ www.trifacta.com.

Fig. 2. Fragment of the myGrid Ontology.

*the corresponding value obtained as a result of the module invocation. By examining such a data example, a domain expert will be able to understand that the **GetRecord** module retrieves the protein record that corresponds to the accession number given as input.*

2.1 Data Example Generation

Enumerating all possible data examples that can be used to describe a given module may be expensive or impossible since the domains of input and output parameters can be large or infinite. Moreover, data examples derived in such a manner may be redundant in the sense that multiple data examples are likely to describe the same behavior of the module. A solution that can be used is to create data examples that cover the classes of behavior of the module in question, and then construct data examples that cover the classes identified. When the modules are white boxes, then their specification can be utilized to specify the classes of behavior and generate the data examples that cover each class (see e.g., [1]). If, on the other hand, the modules are black boxes and their specification is not accessible, then a heuristic such as the one described in [2] can be utilized. To make our paper self-contained, we will describe the solution presented in [2] for generating data examples. We stress, however, that our approach for module discovery is not confined to modules described using the approach presented in [2]. Instead, it can be applied to potentially any module repository where the modules are described using data examples that are annotated with semantic domain concepts.

Using the solution proposed in [2], to construct data examples that characterize the behavior of a module m , the domain of its input i is divided into partitions, $p_1, p_2, \dots, p_n$. The partitioning is performed in a way to cover all classes of behavior of m . For each partition p_i , a data example δ is constructed such that the value of the input parameter in δ belongs to the partition p_i . A source of information that is used for partitioning is the semantic annotations used to describe module parameters. Indeed, the input and output parameters of many scientific modules are annotated using concepts from domain ontologies [7]. In its simple form, an ontology can be viewed as a hierarchy of concepts. For example, Fig. 2 illustrates a fragment of the myGrid domain ontology used for annotating the inputs and output parameters of bioinformatics modules [4]. The concepts are connected together using the subsumption relationship, e.g., **ProteinSequence**

is a sub-concept of `BiologicalSequence`, which we write using the following notation: `ProtSequence` $\sqsubseteq$ `BioSequence`. Such a hierarchy of concepts can be used to partition the domain of parameters.

To generate data examples that characterize the behavior of a module m , m is probed using input instances from a pool, the instances of which cover the concepts of the ontology used for annotations. The retrospective provenance obtained as a result of the module's executions are then used to construct data examples. In doing so, only module executions that terminates without issues (that is without raising any exception) are utilized to construct data examples for m . For more details on this operation, the reader is referred to [2].

3 Module Discovery

To discover a module, a user can provide data examples that characterize the module s/he had in mind. However, specifying data examples that characterize the desired module can be time-consuming, since the user needs to construct the data examples by hand. We present in this section a method that allows users to discover modules by simply providing feedback on a list of data examples they are presented with.

3.1 Feedback-Based Discovery of Scientific Modules

To identify the modules that meet his/her needs, the user starts by specifying the semantic domains and the structural types of the inputs and outputs of the modules s/he wishes to locate. The modules with inputs and outputs that are compatible with the specified semantic domains and structural types are then located. Consider, for example, that the user is interested in locating a module that consumes input values that belong to the semantic domain c_i and structural type t_i , and produces output values that belong to the semantic domain c_o and structural type t_o . A module m meets such a query if it has an input (resp. output) with a semantic domain and structural type that are equivalent to or subsumed by c_i and t_i (resp. c_o and t_o). Specifically, the set of modules that meet those criteria can be specified by the following set comprehension:

$$\{m \text{ s.t. } (\exists i \in \text{inputs}(m), (\text{sem}(i) \sqsubseteq c_i) \wedge (\text{str}(i) \sqsubseteq t_i)) \\ \wedge (\exists o \in \text{outputs}(m), (\text{sem}(o) \sqsubseteq c_o) \wedge (\text{str}(o) \sqsubseteq t_o))\}$$

It is likely that not all the modules retrieved based on the semantic domain of input and output parameters perform the task that is expected by the user. Because of this, we refer to such modules using the term *candidate modules*.

To identify the candidate module(s) that perform the task expected by the user, the data examples characterizing candidate modules are displayed to the user. The user then examines the data examples and specifies the ones that meet the expectations, and the ones that do not. To do so, the user provides feedback instances. A feedback instance uf is used to annotate a data example, and can be defined by the following pair $uf = \langle \delta, \text{expected} \rangle$, where δ denotes the data

Data examples			User feedback	
	protein name	accession	expected	unexpected
δ_1	Chorion protein S36	CH36_CERCA	X	
δ_2	Zinc metalloproteinase	VMDM_VIRST		X

Fig. 3. Data examples and user feedback.

example annotated by the feedback instance uf , and **expected** is a boolean that is true if δ is expected, i.e., compatible with the requirements of the user who supplied uf , and false, if it is unexpected.

3.2 Incremental Ranking of Candidate Modules

The discovery strategy we have just described can be effective when the number of candidate modules and the number of data examples characterizing each candidate are small. If the number of candidate modules to be annotated and/or the number of data examples used for their characterization are large, then the user may need to provide a large amount of feedback before locating the desired module among the candidates. Moreover, there is no guarantee that the set of candidates is complete in the sense that it contains a module that implements the behavior that meets user requirements. Therefore, the user may have to annotate a (possibly) large number of data examples only to find out that none of the candidates meet the requirements. Because of the above limitations, we set out to develop a second discovery strategy with the following properties:

1. **Ranking candidate modules:** Instead of simply labeling candidate modules as suitable or not to user requirements, they are ranked based on metrics that are estimated given the feedback supplied by the user, to measure their fitness to requirements. In the absence of candidates that meet the exact requirements of users, ranking allows the user to identify the modules that best meet the requirements among the candidate modules.
2. **Incrementality:** The user does not have to provide feedback annotating every data example characterizing the candidate modules before being presented with the modules that best meet the requirements. Instead, given feedback supplied by the user to annotate a subset of the data examples, the candidate modules are ranked and the obtained list of candidates is shown to the user. The list of candidates is incrementally revisited as more feedback instances are supplied by the user.
3. **Learning feedback:** To reduce the cost in terms of the amount of feedback that the user needs to provide to locate suitable modules, new feedback instances annotating data examples that the user did not examine are inferred based on existing feedback that the user supplied to annotate other data examples.

Ranking Candidate Modules. To be able to rank candidate modules, we adapt the notions of precision and recall [10] that are used in information retrieval, to estimate the fitness of a module to user requirements based on the feedback supplied by the user. Consider that the user provided the feedback instances $\mathbf{UF}$ to annotate some (not necessarily all) data examples that characterize the candidate modules. We define the precision of a candidate module, $\mathbf{m}$, relative to the feedback instances in $\mathbf{UF}$ as the ratio of the number of true positives of $\mathbf{m}$ given $\mathbf{UF}$ to the sum of true positives and false positives of $\mathbf{m}$ given the feedback instances in $\mathbf{UF}$. That is:

$$\text{precision}(\mathbf{m}, \mathbf{UF}) = \frac{|\text{tp}(\mathbf{m}, \mathbf{UF})|}{|\text{tp}(\mathbf{m}, \mathbf{UF}) + \text{fp}(\mathbf{m}, \mathbf{UF})|}$$

where $\text{tp}(\mathbf{m}, \mathbf{UF})$ (resp. $\text{fp}(\mathbf{m}, \mathbf{UF})$) is the set of data examples describing the module $\mathbf{m}$, and that are annotated as expected (resp. unexpected) by feedback instances in $\mathbf{UF}$, i.e:

$$\begin{aligned}\text{tp}(\mathbf{m}, \mathbf{UF}) &= \{\delta \in \Delta(\mathbf{m}) \text{ s.t. } \langle \delta, \text{true} \rangle \in \mathbf{UF}\} \\ \text{fp}(\mathbf{m}, \mathbf{UF}) &= \{\delta \in \Delta(\mathbf{m}) \text{ s.t. } \langle \delta, \text{false} \rangle \in \mathbf{UF}\}\end{aligned}$$

Ranking based on precision only may not be enough: a module may be associated with the maximum precision of 1, i.e., all its data examples are true positives, and yet it may not implement all the classes of behavior expected by the user. Recall can be used to identify such modules. The recall of a module $\mathbf{m}$ relative to the feedback instances in $\mathbf{UF}$ can be defined as the ratio of the number of true positives of $\mathbf{m}$ given $\mathbf{UF}$ to the sum of true positives and false negatives of $\mathbf{m}$ given the feedback instances in $\mathbf{UF}$. That is:

$$\text{recall}(\mathbf{m}, \mathbf{UF}) = \frac{|\text{tp}(\mathbf{m}, \mathbf{UF})|}{|\text{tp}(\mathbf{m}, \mathbf{UF}) + \text{fn}(\mathbf{m}, \mathbf{UF})|}$$

where $\text{fn}(\mathbf{m}, \mathbf{UF})$ denotes the false negatives of $\mathbf{m}$ given the feedback instances in $\mathbf{UF}$. To illustrate what we mean by a false negative data example, consider δ' a data example that is the user annotated as expected. δ' is a false negative for the module $\mathbf{m}$ if when invoked using the input values specified by $\mathbf{n}\delta'$, the module $\mathbf{m}$ returns output values that are different from the output values specified by δ' .

$$\text{fn}(\mathbf{m}, \mathbf{UF}) = \{\delta \text{ s.t.}, \langle \delta, \text{true} \rangle \in \mathbf{UF} \wedge \text{not match}(\text{invocation}(\mathbf{m}, \delta.\mathbf{I}).0, \delta.0)\}$$

where $\text{invocation}(\mathbf{m}, \delta.\mathbf{I}).0$ denotes the output values delivered by the module $\mathbf{m}$ when it is invoked using the input values specified by the data example δ .

$\text{match}(\text{invocation}(\mathbf{m}, \delta.\mathbf{I}).0, \delta.0)$ is a boolean that is true if the output values delivered by the invocation of the module $\mathbf{m}$ are the same as the output values specified by the data example δ .

To rank candidate modules, we use the F-score, which combines precision and recall using the harmonic mean as illustrated below. The module associated with the highest F-measure is the candidate that best meets user requirements given the feedback instances in $\mathbf{UF}$.

$$\mathbf{F}(\mathbf{m}, \mathbf{UF}) = \frac{2 \times \text{precision}(\mathbf{m}, \mathbf{UF}) \times \text{recall}(\mathbf{m}, \mathbf{UF})}{\text{precision}(\mathbf{m}, \mathbf{UF}) + \text{recall}(\mathbf{m}, \mathbf{UF})}$$

Learning Feedback. Notice that the method for identifying false negatives of candidate modules that we have just described can be computationally expensive. In particular, every candidate module m may need to be invoked using all data examples that are not used to characterize m and that are labeled as expected by the user, i.e., the data examples in $\text{expected}(UF) - \text{tp}(m, UF)$.

To overcome the above problem, we adopt an approach that not only reduces the number of times a candidate module needs to be invoked (using known expected data examples) to identify false negatives, but also allows learning new feedback instances that the user would give on unannotated data examples based on existing feedback instances. To illustrate the approach we adopt for this purpose, consider the candidate module `getAccession` and `getAccessionOfSimilarProtein` (see Fig. 3). These two modules consume a protein name and output a protein accession, and are characterized by one data example each because the concepts `ProteinName` and `ProteinAccession` are leaf nodes in the ontology used for annotation. The feedback supplied by the user to annotate the data examples δ_1 and δ_2 illustrated in Fig. 3 shows that δ_1 is expected and δ_2 is unexpected. Therefore, δ_1 is a true positive for the module `getAccession`, and δ_2 is a false positive for the module `getAccessionOfSimilarProtein`. Now, to know whether δ_1 is a false negative for the module `getAccessionOfSimilarProtein`, we will need to invoke `getAccessionOfSimilarProtein` using the input value specified in δ_1 , i.e., Chorion protein S36.

Intuition Behind Feedback Learning. Using the solution that we adopt, we do not need to invoke `getAccessionOfSimilarProtein`. To do so, we slightly modify the process by which data examples are constructed to cover the partitions of input parameters presented in [2] and overviewed in Sect. 2.1. Specifically, when selecting input values for data examples to cover a given partition, i.e., semantic domain, c , the same input value v (in c) is used in all those data examples. For example, using this method, the data examples used to characterize the two modules `getAccession` and `getAccessionOfSimilarProtein` will have the same input value. Figure 4 illustrates the data examples δ_1 and δ_3 specified using this method to characterize such modules.

Consider that the user supplies the feedback instance annotating the data example δ_1 as expected (see Fig. 4). Given this feedback instance, we do not have to invoke the module `getAccessionOfSimilarProtein` using the input value specified in δ_1 to know if δ_1 is a false negative for `getAccessionOfSimilarProtein`. Indeed, the data example δ_3 shows that the output produced by `getAccessionOfSimilarProtein` using the same input value as that used in δ_1 . Given that the output values of δ_1 and δ_3 are different, we can make the following inferences: i) δ_1 is a false negative for `getAccessionOfSimilarProtein`, moreover, ii) δ_3 is unexpected, and is, therefore, a false positive for `getAccessionOfSimilarProtein`. This last inference can be made because the modules that we consider are deterministic. Therefore, the fact that δ_1 is expected implies that δ_3 is unexpected. Note that if δ_3 had

Fig. 4. Example illustrating feedback inference.

the same output value as δ_1 , then we would have inferred that δ_3 is expected and is, therefore, a true positive for `getAccessionForSimilarProtein`.

4 Concluding Remarks

To assess the performance of the discovery strategy described in the previous section, we ran an experiment to identify the amount of feedback required to detect the modules that are relevant to users' needs. We also examined the error in the F-score estimates computed for candidate modules based on user feedback. To perform a systematic sweep of the parameters of the experiment, we use a synthetic dataset that we created for this purpose. We also used real-world bioinformatic modules.

The result of this experiment showed that users can effectively discover scientific modules using a small number of feedback instances. A particularly interesting result that we empirically showed is that the number of feedback instances that the user needs to provide to identify the module that meets the requirement, and more generally a ranking that meets his/her expectations, is small even in the cases where the number of data examples describing the behavior of the modules is large.

References

1. Alexe, B., Cate, B.T., Kolaitis, P.G., Tan, W.C.: Characterizing schema mappings via data examples. *ACM Trans. Database Syst.* **36**(4), 23:1–23:48 (2011)
2. Belhajjame, K.: Annotating the behavior of scientific modules using data examples: a practical approach. In: *EDBT*, pp. 726–737. OpenProceedings.org (2014)
3. Ebaid, A., et al.: NADEEF: a generalized data cleaning system. *PVLDB* **6**(12), 1218–1221 (2013)
4. Goble, C., et al.: BioCatalogue: a curated web service registry for the life science community. In: *Microsoft eScience Conference* (2008)
5. Gruber, T.: *Ontology*. In: *Encyclopedia of Database Systems* (2009)
6. Konstantinou, N., et al.: The VADA architecture for cost-effective data wrangling. In: *SIGMOD Conference*, pp. 1599–1602. ACM (2017)
7. Kuroпка, D., Tröger, P., Staab, S., Weske, M. (eds.): *Semantic Service Provisioning*. Springer, Heidelberg (2008). <https://doi.org/10.1007/978-3-540-78617-7>

8. Stonebraker, M., et al.: Data curation at scale: the data tamer system. In: CIDR (2013). www.cidrdb.org
9. Studer, R., Grimm, S., Abecker, A. (eds.): Semantic Web Services, Concepts, Technologies, and Applications. Springer, Berlin (2007). <https://doi.org/10.1007/3-540-70894-4>
10. van Rijsbergen, C.J.: Information Retrieval. Butterworth, London (1979)