Abstraction Layer for Data Cleaning Tools

Milad Abbaszadeh Jahromi[1], **Mohammad Mahdavi**[1], **Ziawsch Abedjan**[1]

[1]Technische Universität Berlin

{milad.abbaszadehjahromi, mahdavilahijani, abdejan}@tu-berlin.de

Abstract. *Various data cleaning tools have been developed so far in data cleaning community. These tools are usually pointwise softwares that focus on specific types of data quality problems. Although combining these data cleaning tools together seems a promising idea, it is a hard task due to the heterogeneity of these tools. Each of these data cleaning tools has been written in different programming languages and has its own input/output format.*
In this document, we introduce our abstraction layer that hides these underlying heterogeneities from the user's application. The abstraction layer easily installs and configures the tools and provides fixed input/output system for running different data cleaning tools.

1. Introduction

1.1. Motivation

Data cleaning is the broad family of approaches that are aiming at improving data quality. There are different data cleaning tools that have been developed so far. Although combining these tools seems promising, heterogeneity and dispersion of these tools make it difficult to combine tools into one application. Each tool has been developed by a different people, in different programming languages. Furthermore, they usually have different installation processes and input/output systems. For a user that wants to combine these tools in a unique application, combining these heterogeneous data cleaning tools could be difficult, time consuming, and frustrating.

In this document, we introduce our abstract layer that provides an easy-to-use framework for the user to work with different data cleaning tools. In particular, our abstraction layer provides simple APIs for installation of tools and running them on the datasets.

1.2. Contributions

Our abstraction layer provides the following contributions:

- Simple and guided process for installing different data cleaning tools.
- Easy-to-use framework to run different data cleaning tools by fixed input/output formats.

1.3. Architecture

Figure 1 shows the position of the abstraction layer between the user's application and data cleaning tools. The user's application communicates with the abstraction layer in fixed input/output formats, regardless of the tool that application aims at running. Internally, the abstraction layer contains several wrappers for communicating with different data cleaning tools. Therefore, the heterogeneity of tools would be hidden for the user's application.

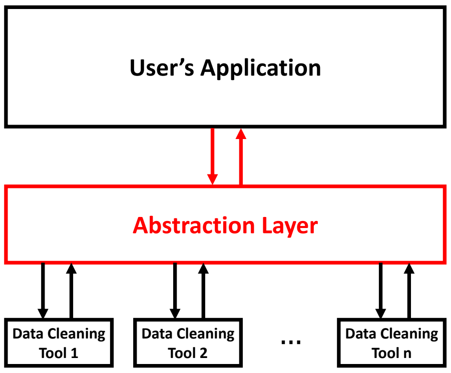

Figure 1. The position of the abstraction layer between the user's application and data cleaning tools.

1.4. Overview

The rest of this document is organized as follows. In Section 2, we explain how the user can install the module. In Section 3, we introduce the input/output format of the abstraction layer and provides some examples.

2. Installation

2.1. Prerequisites

To install the module, you need first to install the followings:

- Linux (Ubuntu/Debian recommended)
- Python 2.7
- Oracle Java 1.8
- Apache Ant 1.8.2+
- PostgreSQL 9.2+

2.2. Setup

After cloning the project, to install and configure the underlying data cleaning tools, it is enough to call the following API function:

```
install_tools()
```

This method installs and configures all the tools.

3. Start to Work

To communicate to the abstraction layer, the user's application must call the following API function:

```
run_output = abstraction_layer(run_input)
```

This method takes a data cleaning job as input and returns the subsequent results as output. The formats of input/output objects are fixed. We explain these formats in the following subsections.

3.1. Input

To call the abstraction layer, the user needs to provide a data cleaning job as input. This data cleaning job describe the data cleaning task that the abstraction layer should do for the user's application. The general template of the data cleaning job is as follows:

```
run_input = {
  "dataset": {
    "type": "dataset type",
    "param": ["list of parameters wrt. the dataset type"]
  },
  "tool": {
    "name": "tool name",
    "param": ["list of parameters wrt. the tool name"]
  }
}
```

In practice, the data cleaning job is a JSON text that contains the following information:

- **dataset**. This part of the input configuration is responsible to describe the input dataset. It consists of two options.
  - **type**. This option specifies the type of dataset. Currently, the module only supports *csv* datasets.
  - **param**. This option specifies the list of parameters that are needed for accessing the dataset. For csv datasets, this list contains only the path of dataset.
- **tool**. The second part of the input configuration is responsible to describe the data cleaning tool that user aims at running on the input dataset. It consists of two options.
  - **name**. This option specifies the name of the tool. Currently, the module supports *dboost* [3], *nadeef* [2], *openrefine* [4], and *katara* [1] tools.
  - **param**. This option specifies the list of parameters that are needed for running the selected tool.

Example 1: Running dBoost.

```
run_input = {
  "dataset": {
    "type": "csv",
    "param": ["datasets/sample.csv"]
  },
  "tool": {
    "name": "dboost",
    "param": ["gaussian", "1"]
  }
}
```

Example 2: Running NADEEF.

```
run_input = {
  "dataset": {
    "type": "csv",
    "param": ["datasets/sample.csv"]
  },
  "tool": {
    "name": "nadeef",
    "param": [["title", "brand_name"]]
  }
}
```

3.2. Output

After calling the abstraction layer with proper input data cleaning job, it generates a list as output. This list consists of the cells in dataset, that have been marked as data errors along with the value that the data cleaning tool suggests as correction. The general template of the output list is as the follows:

```
run_output = [(i1, j1, v1), (i2, j2, v2),...]
```

Each element in the output list contains exactly three values referring to one specific cell in the dataset:

- **Row number (i).** The row number of detected cell.
- **Column number (j)** The column number of detected cell.
- **Suggested value (v)**. The suggested value for this cell.

References

[1] X. Chu, J. Morcos, I. F. Ilyas, M. Ouzzani, P. Papotti, N. Tang, and Y. Ye. Katara: A data cleaning system powered by knowledge bases and crowdsourcing. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1247–1261. ACM, 2015.

[2] M. Dallachiesa, A. Ebaid, A. Eldawy, A. Elmagarmid, I. F. Ilyas, M. Ouzzani, and N. Tang. Nadeef: a commodity data cleaning system. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 541–552. ACM, 2013.

[3] C. Pit-Claudel, Z. Mariet, R. Harding, and S. Madden. Outlier detection in heterogeneous datasets using automatic tuple expansion. Technical Report MIT-CSAIL-TR-2016-002, CSAIL, MIT, 32 Vassar Street, Cambridge MA 02139, February 2016.

[4] R. Verborgh and M. De Wilde. *Using OpenRefine*. Packt Publishing Ltd, 2013.