Abstraction Layer for Data Cleaning Tools

Milad Abbaszadeh Jahromi[1]

[1]Technische Universität Berlin

Milad.abbaszadehjahromi@tu-berlin.de

Abstract. *This is the abstract... Mohammad: leave abstract. i will write it.*

1. Introduction

1.1. Motivation

Despite the increase amount of data which is generating every single second[1] and the value has made out of that, the data quality is even more important in compare to several decade ago. One of the ways that helps this approach is data cleaning.Although there aren't many tools that works in this area,heterogeneity and Dispersion of each tool make the process harder.Furthermore,each tool has different installation,input and output structure that can be time consuming and be catastrophic for users and forced them to stick to each tool structure that isn't desirable for most of the experts. This report represents the abstract layer which can help users to avoid from conflicting with the tools for cleaning purpose and help them for unique and constant input and output which can be really remarkable to save time for experts and let them to work with single and common input and output for all tools that covered by abstract layer.

Mohammad: the motivation can be so much better and stronger. talk about different tools that have heterogeneity in terms of language, installation, input/output system and so on and the user difficulties.

1.2. Contributions

experts of cleaning, most of the time use cleaning tools and they don't like to get involved in installation and command usage which is required for running, you can simply use our abstract layer. In other words because of the following reasons experts should use this abstract layer:

- Simple install
- Easy to use
- Hide installation of each tool
- constant Input
- constant output
- constant instructed command

With this layer data has been sent as result, the Suspicious elements come up with constant type and format which is generated as a output for each tool.

Mohammad: talk about how this module can address the so-called heterogeneities for the user's application. use bullet list just like the other papers' contributions. talk about user instead of "you".

1.3. Architecture

The abstract layer located as part of user code in real world like common dictionary and it has utilized as Connector between the user's application and each tool which is consumed for cleaning purpose[figure 1] and try to hide the levels of communicating with tools.

Mohammad: in this subsection talk about the big picture of the architecture and the position of the abstraction layer in the user system. I have added a proper figure. you write the texts.

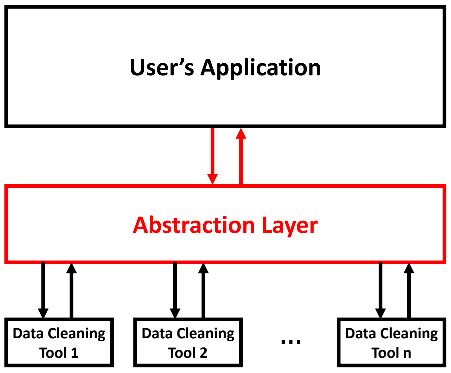

Figure 1. The position of the abstraction layer in the architecture of the user's system.

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 has introduced about input and output format and then some example are generated for better understanding.

Mohammad: complete this subsection

2. Installation

2.1. Prerequisites

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

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

2.2. Setup

the designed library has two main part, first part check the installation on the tools that supported by abstract layer and if it is needed the following tools will be install automatically and store in respect place that the layer is located. For communicate and execute the installation the respect function is Considered:

```
install_tools()
```

for working with second part of layer the function with the appropriate input should call that is introduced in the following section.

Mohammad: talk about how the user can use the installation function in the first place to installs and configures the tools. introduce the API function that installs and configures the tools.

3. Start to Work

3.1. Input

For starting with abstract layer, required to make input configuration with the rules that is introduced by detail in the following. Input configuration really is a dictionary that make by users and give to abstract layer as input and let functions to starts.This dictionary has two keys, the first one clarify type of the file and location of the file as path. In fact, for now the layer only support the files that generate with csv format but we will keep going to add other formats for future.

Mohammad: before introducing the template of your input configuration (look below), first start to talk generally about: (1) proper api function. (2) define the argument that it needs (input configuration, data cleaning job, or whatever you call it), then you have the following codes:

```
run_input = {
  "dataset": {
    "type": "csv",
    "param": ["the/address/of/your/file.csv"]
  },
  "tool": {
    "name": "tool name",
    "param": ["list of parameters wrt. to the tool name"]
  }
}
```

Mohammad: after the general template of the input object, start to explain each key and its proper values in a nested list:

- **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 consist on the type of tool that user have plan to use it and parameters that each tool need for running.
  - **name**. In this section the user clarify name of the tool which is want to use it.Currently ”Nadeef” and ”Dboost” covered by abstract layer.
  - **param** This option specifies the parameters and entry of each tool regarding to the tool that has been chosen in the section before for tool’s name.

Example for Nadeef:

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

Example for Dboost:

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

3.2. Output

Regardless of data type and tools that is selected by user for data cleaning job, the abstract layer return constant output.

- **Return:** user will receive one list as output that suspicious elements by row,column and old value return to user after execution.
  [row, column, suspicions value]

  Mohammad: just like the input subsection, write in this subsection.

References

[1] Dallachiesa, M., Ebaid, A., Eldawy, A., Elmagarmid, A., Ilyas, I. F., Ouzzani, M., Tang, N. (2013, June). NADEEF: a commodity data cleaning system. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (pp. 541-552). ACM.