Intelligent tool for visual data analysis

Bachelor's Thesis

Jorge Osés Grijalba

Double Major in Mathematics and Computer Science
Computer Science Faculty
Complutense University of Madrid

March 2019

Documento maquetado con TEXIS v.1.0.

Intelligent tool for visual data analysis

Texto

Dirigida por los Doctores
Belén Díaz Agudo, Juan Antonio Recio García

Double Major in Mathematics and Computer Science
Computer Science Faculty
Complutense University of Madrid

March 2019

Abstract

There is light at the end of the tunnel...
hopefully it's not a freight train.
M. Carey

This document reflects my Bachelor's Thesis corresponding to the Double Degree in Mathematics and Computer Science, developed within the area of intelligent data analytics and 'Case Based Reasoning'. During the progress of the project, the principles applicable in any environment of data processing and the science behind it are explained generally and aimed to be usable in any kind of context by any user provided the right format of data. Nowadays, highly heterogeneous data collection and processing methods are employed in all industries, however the techniques employed to get useful information out of the data usually have a generalistic aim, and the work relevant to the field itself is often done manually. In this work we aim to provide an automated way to analyze information while taking into account information and techniques relevant to the field of the analysis. The objective of this Degree's Final Project is the development of a prototype capable of carrying this analysis while being able to learn based on user input. As a Proof of Concept, we have included several medical domains with each one having developed specific methods and techniques for them. To serve as a base for this analysis, we have also developed a system for storing, loading and analyzing the information of the domain and the information provided by the user. This system will be the backbone of our architecture and enable the Case Based Reasoning analysis to function correctly in very different situations, providing the metrics and functions needed for every case.

Contents

Chapter 1

Introduction

Abstract: In this chapter we outline the thesis, state what our objective is and what we hope to achieve and list the programming libraries, techniques and methods used to achieve our goal.

1.1 The Initial Problem

The need for performing analysis on large amounts of data to get very precise and specific information is becoming more and more present everyday in the jobs of data analysts and scientists in every field of work.

Large amounts of time are wasted on repetitive tasks such as data wrangling, data transforming and the generation of tailored reports or collections of information with different objectives for diverse profiles with varying degrees of expertise.

These reports are usually formed by a piece of text acompanied by some graphs.

It is very common that from these huge amounts of data we want to extract some precise and relevant information to be presented to someone.

These reports have a process behind them that entails the filtering, transformation and selection of the relevant information that will finally be part of the report.

Here we have two questions to answer. First, we must choose what information to present, which is equivalent to choosing what information from the almost unlimited attributes that our data has is relevant to the user.

It is clear that this has an objective part, in the sense that it is first and foremost a matter of which data is relevant within a certain domain of knowledge and a certain set of metrics, but it also has a subjective side, in the sense that it's not the same to present medical information to a patient or to present it to a doctor.

The second answer, and perhaps the most subjective is how to present it. This second decision is related to things like choosing a type of graph, its colors, the font of the text, the words used... and almost an infinite list of subjective choices that build upon the previous more objective selection of information to form the final concept of a report of a piece of information.

1.2 Solution Proposed

We believe that there is no unique formula to generating each report, because it would require the abstraction of very different problems in very different situations and for an almost unlimited variety of users.

Furthermore, what if we have to generate a report for a new user? Could this be similar to other reports presented for other users?

We believe that most of these questions can't be answered by a rigid mathematically formulated system, and are best tackled by a mixed system that combines an objective analysis of the information through a variety of metrics, analysis of correlations, distributions and other objective metrics with a subjective approach that takes the final user into account.

Instead, what we propose is a mixed system in which an expert provides an initial input that signals some of the important aspects of the information, and then a pool of experts validates the subjective way in which an information is presented to end up choosing a default report form to present to a new user of their class.

To start, we categorize the users or people that will be presented with the report into groups. Then, these groups will provide knowledge of the relevant objective information that they're looking for in the data, like what analysis to perform or what values of certain metrics they'd consider to be relevant.

Once this information is fed to the system, it's able to generate reports completing the subjective decisions from semi-random choices from a pool of computer generated graphs, color choices and text based reports.

Then our pool of experts proceeds to validate the best report by a voting system based on an ELO tournament.

The best selected report is then presented as default to users of this class.

Each user will also be able to change the result presented to them in terms of both content (objective) and presentation (subjective), and because the system recognizes individual users it will remember their choices.

Users will also have a profile attached to them containing relevant data to the presentation of the information that will allow us to define a metric of sorts between users to further use this single user customization to influence how information is presented to users of the same class once enough individual inputs have been recorded, possibly substituting the initial ELO based report which will always act as default.

1.3 Overview

The logical structure chosen for the program reflects the need for our tool to be a fully functional agent in and out of itself. We have designed it with a clear divider between a backend capable of storing the information and handling at the lowest possible level, which provides the frontend side with easy methods to get the information it needs, which is then processed taking who is going to look at it into account and then adequately presented to the user.

A cornerstone of the program's functionality is to be able to remember decisions taken by a certain user and to be able to compare new data to old data of its kind.

From these two necessities it is natural to consider some kind of identification system for our datasets, as automated as possible so it needs minimum user input and remains independent of the use case.

For our program, if two datasets contain the exact same set of column names then they are considered to be comparable to each other, and every information stored about this kind of datasets will carry an identifier with the column names.

From here onwards, the term 'domain' shall refer to information coming from the same kind of dataset.

1.4 Workflow

When a new dataset doesn't match any previous knowledge, our program automatically creates a new representation for these datasets which is stored along the others. If it detects a matching JSON with knowledge o its domain it loads that instead.

Each representation of a domain stores data such as how many datasets have been loaded and a number of stats for each dataset and its columns depending on its types which will be specified later.

Also, each domain has a number of 'profiles' associated which correspond to *who* this data is associated with. These profiles contain both historical data of the specific owner of the data (in our practical example, the patient data), and who will watch the report generated by this program, that is, the user of the program.

The information that we're using will be stored in a specific JSON format for each kind that will be specified in chapter 2.

When a dataset is introduced, the program loads the previous information, analyzes it, compares it and generates relevant information to the user. Then it updates the information with both the results of the analysis and user provided information.

This workflow will be the basic use case of the program for every kind of data.

1.5 Structure

A clear module structure is provided so each module does a task in the workflow.

The main modules on the backend side of our application are the Storage module, the CBRStorage module and the Analysis module.

For the frontend, the logic structure will be split into the Reporter module and the Presentation module.

1.6 Tools

Our programming language of choice has been Python, particulary making use of its class to dictionary representation methods which make the work of manipulating the JSON structures much easier than using more rigid languages.

A public repository has been created at github.com/jorses/tfg, and we have used Jupyter Notebooks for the testing and formation o a prototype which has been then moved into standard Python packages.

Chapter 2

The program structure

Abstract: In this chapter we provide an analysis of the program structure from an abstract point of view.

2.1 The Structure

As we have stated before, our program consists of a series of modules designed to interact with each other and provide the necessary tools to work smoothly regardless of the data being used.

Here we take a look at each functional module, providing a clear abstract picture of what our tool really is.

2.2 Case presentation

The input for our program is really just an user who desires to analyze a certain dataset.

From now on on this chapter, we assume we have previous information for both the domain the dataset belongs to and about the type of user that is trying to analyze it.

We will expand on how the base cases are formed and how we adapt to new profiles and new domains on further chapters, but for now let's say that a process is in place to ensure that the default report is correct enough, the analysis performs its due processes and the result is at least acceptable and relevant to the final user.

2.3 Handling the Information

First and foremost we need a way to store information and to retrieve it effectively. We will store two different kinds of information : objective information about the results of analysis performed on a dataset and information on what information to present and how to present it to the user.

Since the first is related to the domain and applies to all users, and the second concerns both the domain and the user, it makes sense to store them and handle them separately.

Regarding the first kind, we now provide both a way to identify which domain it belongs to and a general description of what it contains, which will be specified later during the technical implementation chapter.

It contains statistics and properties from both columns and datasets, different for different types, as well as how they were measured. Saving how they're measured is important to measure new datasets and to be able to compare metrics effectively.

Both the column specific stats and the dataset stats are subjective to change with each dataset added to the domain, so it is important that we're able to update this information readily and effectively.

It's also important to be able to distinguish between domains. The solution proposed is to use the columns of each dataset as a unique key that identifies a domain, and their objective information will be stored as part of the same object.

In the case of subjective kind, we combine this with a unique user identificator to make a unique key to identify the information.

Each of these structures has parts of the program dedicated to loading this information into the program and updating it when presented with new datasets and storing it overwriting the previous information, creating a dynamic system capable of learning from users and gathering new insights.

2.4 Analyzing the Information

This analysis will concern the objective side only, so we don't need to take the user into account for now.

First a dataset is provided as input for our tool. From this dataset, we retrieve the information of the associated domain so we know how to perform the analysis on this new dataset, as it has to be comparable to the domain information.

So, using the metrics and analysis detailed in the domain information, we perform them on the dataset and compare the results with those of the domain through a series of comparison metrics which will be detailed later.

These results are then condensed into an object containing all the objective data from the comparisons.

This output will be the input for the subjective side of our program, as this information will then be filtered and transformed in a way tailored to the user.

2.5 The Subjective Analysis

This analysis will concern the subjective side, and is to be performed after the previous step has been completed.

From the representation of the objective analysis, we need to perform two tasks.

One is the filtering of such information, selecting the information relevant to the user, and the other is choosing how to present this information to the user.

To perform these two tasks we first load the information we know about the user, which has been stored separately from the objective information as we have previously stated.

We then filter the information through the user provided profile, selecting which comparison is most relevant. Finally we form a report using the graphical information on how to present each bit of data to this user.

2.6 Presentation and Feedback

Once the user has been presented with the report, he's able to make changes to it using a graphical interface. When the report is saved and the user has exited the program, it initializes a shutting down routine on which the two information databases are updated with the changes made by the user (if they existed) and the new information provided by the dataset.

This is the main way our program has of expanding its knowledge, which will then be used to present better reports to the same user and to adapt the information it presents to new users which will be similar to this one.

So we get a double benefit, getting both better results for this user and for all users of the same domain and class each time someone uses our program.

Chapter 3

The Program Implementation : Architecture

Abstract: In this chapter we provide a technical analysis of our tool, examining how it works and what was designed for at a programming level, aswell as its module structure.

3.1 The Program Module Structure

First we need to provide our program with a clear programming structure that allows it to be customizable enough, as it is very important that we're able to add new functionality related to new kinds of analysis, users or datasets.

To make it as customizable as possible, we've divided it in modules that have a clear functionality, with the objective that we're able to change an aspect of the program by changing just one module and not the whole program.

All the code is openly hosted at www.github.com/jorses/tfg.

We will have one module which handles the loading and storing of the objective information about domains, one that does the same for user profiles associated to each domain, one that runs an analysis on the dataset and compares it to the domain, and one that puts everything together to generate a report which will be then presented to the user through the frontend module, which handles interactions with the user.

So our hierarchy goes as follows, the frontend module takes input from the user regarding its profile (input done through a dictionary-like object in the proof of concept) and a path to the dataset that is to be analyzed.

This module then relays the information to the two modules that handle information storage.

The storage module will read the dataset, as a CSV file in the proof of concept, read its columns and search its database for a matching domain information.

In a more advanced version this search would be performed through a lookup on a non relational database like MongoDB, but for the proof of concept it looks in a folder where it stores the domain information objects as JSONs.

When it finds one that matches the column names, it loads the object into a python dictionary and sends it to the analysis module along with the new dataset.

The analysis module then performs an analysis and generates a proto-report with all the objective information about the comparison between the dataset analysis and the domain historical analysis, then passes this along to the report generation module.

In parallel, the module that handles the profile will load it in a way similar to the storage module, then pass it along to the report generation module.

When the report generation module gets all the needed information uses the user-generated profile to "filter" the objective results (as explained later in detail), and extract from them the final info that will then be presented to the user through the frontend module.

The user is able to make changes to this final report (through the console in the proof of concept), and before closing the program

All of this enables the CBR process which is explained in chapters 5 and 6.

3.2 Non relational databases and the JSON structure

For the proof of concept implementation developed in this thesis, we've used the JSON structure to handle our information, to write it to a file system and to read it later.

JSON, short for JavaScript Object Notation is a file format that uses text readable by humans to write attribute value pairs and serializable values.

It is a a very common data format used for general storage of information, and will allow us to do a quick implementation capable of handling enough domains and profiles to test our program, and will allow us to later transfer this information to a non-relational database system such as MongoDB.

Python also allows us to quickly load this kind of object into its native dictionary type, making it easy for our modules to load, manipulate and store the information.

A non relational database is one that does not use the tabular schema of rows and columns found in most typical databases like SQL.

Our need for such a database comes from the fact that not every domain will have the same structure of knowledge for its analysis, with the possibilities of combinations being almost infinite and thus negating any approach to putting it inside a tabular schema.

Inside these databases data may be kept as JSON documents, which is the format that we've chosen.

The proposed database for the final model is MongoDB.

MongoDB is a non relational database program, which is also cross-platform and document oriented. It works with objects very similar to JSON, and is licensed under the Server Side Public License (SSPL).

Aside from this, it's easily managed through Python and has native packages to communicate with it, which would make our life easier when implementing the full program.

All of this provides us with a solution catered to our needs, which can be scalated easily if needed into a production ready state, capable of dealing with large amounts of data and providing a real solution to real users.

Every module listed below corresponds to a Python class, and has a number of methods, functions and dictionaries associated to it to perform its function.

3.3 The Objective Storage Module

This module is capable of reading and updating the information stored about the objective analysis of past domains.

It sits at the lowest point in the hierarchy, as it only does as commanded and is usually handled by the other modules and used as a mere tool to get information, similar to the profile storage module.

Both modules are derived from a more basic Python class, called just Storage, from which they inherit the methods responsible from loading and storing files (or updating and loading from the database, in the scalable version).

This class adds to these functions more advanced tools to deal with the domain information and load it into a dictionary-like object that will then be able to be easily handled by the analysis module, translating information stored in plain text like function names to its equivalent variables and Python dictionaries in the program, using the dictionaries which it has as an attribute.

This process is reversed before writing the information back to disk in a format that can be condensed to JSON documents, mainly turning every function name and other serializable objects back into strings to be stored.

3.4 The Profile Storage Module

Another kind of information is stored about the domains. This is the information concerning the human side of things, that is, how to interpret these stats and turn them into something that humans with different levels of familiarity can understand.

To do this, we provide use another storage class that will contain human-relevant data that will modify the objective comparison delivered by the analysis module.

A system of profiles is added to the object itself, inside a "profiles" key. The domain associated is clear as they share the same "attributelist" identification system.

The information contained in each one of these "profiles" serves two different purposes. It provides customizable elements of how the data will be presented to the user, and it keeps an historical record of this user's dataset results (similar to the one in the domain storage) making a historical following of a profile possible.

For each domain, there's a default profile. This provides a way to present the data when no previous knowledge of the profile is available. The automated processes of obtaining this profile and tuning the existing profiles from user feedback will be explained in further chapters.

3.5 The Comparison Metrics

The purpose of these functions is to provide a way to measure the properties of a given dataset or knowledge domain.

We have to note first that not every kind of possible metric has been added to the program, but we've instead added enough to cover the most common types of analysis, mainly detecting distributions, most frequent values, and calculating a series of common metrics over numerical columns.

However, adding a new metric is as simple as providing it with a name, an associated function which fits into the types of one of the metrics listed below, and adding the pair of name,function to a dictionary present in the code.

The rest of the program will behave exactly the same, thus fittinng the design principle mentioned before of being able to expand the program quickly and efficiently.

We can categorize them as follows:

First the "measurement" metrics, used to get the information of a single dataset or domain.

- Dataset Metrics : they concern the dataset as a whole, like number of rows with missing values. dm :: (ds) –> num

- Single Column Metrics : they concern a certain column, and are based on the type of the column. For numerical columns we will have things like median, averages, deviations, distributions... For categorical columns we'll work with frequencies and things of the sort. scm :: (col) –> num
- Multiple Column Metrics : we will be looking for correlations and things of that sort. scm :: (col,col) –> num Time based metrics will be defined from this construct.

We will also have "comparison" metrics, used to compare datasets against their domains. These metrics will compare the output of two measurement metrics, both will have to spawn from the same function.

- compm :: (metric) –> num

Note that these metrics are not to provide "meaning" or any human-readable input, nor to be inherently comparable between each other outside of a framework of understanding of the domain (metric importance).

A mean to convert these machine cold metrics into human understanding will be provided in further modules. For now, we're not taking humans into account.

3.6 The Analysis Module

The analysis module will receive a dataset, use the Storage module to load its information, then analyze the dataset, which generates a similar object to the domain json, then producing a comparison of both.

The main methods for this module are getstats , getcolumnstats and getdatasetstats .

The getstats method is just a wrapper for the other two, calls them both and stores its results inside the Analyzer class.

Both getcolumnstats and getdatasetstats compute the statistics for the given dataset. If there is previous knowledge of the domain, the stats that appear there are computed for the domain. If not, a standard set of frequencies for categorical values and medians and distributions for numerical values are calculated and used to populate the stats object.

The most important method is getanalysis . Once the stats for the datasets have been generated, if there's previous knowledge available the class runs an analysis comparing the metrics of the two, and generating an object with the result. For this to happen, each metric defined for the dataset must have an associated comparison metric .

If there is no previous knowledge then the dataset stats are passed along to the reporter with a field indicating that there was no previous knowledge.

In any case, at this level we've already filtered what is relevant and what is not from the comparison.

3.7 The Report Generation Module

This module stands at the edge between the backend and the frontend. It receives the information from the comparison between the dataset and the domain knowledge and extracts the relevant profile information.

Once this is done, it uses both to generate a report with all the information from both sides. The user-relevant info will modify what is shown and how that is shown, changing the graphical elements according to the user so the frontend modules are able to be logic-free.

It is able to directly modify the profile information by the proxy methods modify and savehumaninfo . Its main method, generate , will create and populate an attribute within itself called report.

This method is called when the class itself is generated but the report can be modified as any Python attribute if needed.

At this point, we have an object that represents the dataset compared to the historical data and data about the profile associated with the user. This information, however, is in the form of a JSON object and is not really human-readable. The job of the frontend modules is to take this information and turn it into something easy to understand.

3.8 The Frontend Module

The purpose of this module is to handle the user program interaction. It sits atop of the program hierarchy, being able to use all the other modules as it sees fit.

Its role begins and the very beginning of the program life cycle and ends at the same time the user decides to exit.

Its functionality is based on the principles of minimalistic design and the user being able to interact with every element of the presented report.

For our proof of concept, a more simple design has been put in place. Instead of clicking on the elements, the user is able to interact with the program through the console.

However, all the functionality is present, as the users can input the information the program requests to generate the profile, and then change the report presented to them using commands too.

When the program is closed the frontend module passes the modified report to the information storage modules, with both the updated objective information and the updated subjective information.

Chapter 4

Seed Cases and ELO Tournament

Abstract: In this chapter we outline and explain the our techniques used to ensure new users and domains have a valid starting point from which users can productively use the tool.

4.1 Motivation

Our objective for this chapter will be to provide details on the process chosen to develop a solid knowledge basis on which we can build a use case for our program.

What we mean by this basis is the knowledge of which metrics to use to provide a clear picture of the datasets belonging to the domain, as well as which metrics are relevant to a profile and how should they be shown, as with a graph, through a text report, which colors, etc.

We will be using the input of an expert to determine the metrics to use, and will then generate a seed profile based on an ELO tournament with a pool of experts. This will be then affected by user input.

4.2 Case Seeds

The process to establish a frame of knowledge for a certain domain will be initiated by an expert who will provide its associated class, or basic categorization of users, and a brief computer-understood description of what he's interested in.

This knowledge will be used to run a first analysis and generate an array of reports through a semi randomization process, which will be then pooled together as participants of a tournament, and given an initial ELO of 1000.

The final winner of this tournament will be taken as the "seed" for the classification the users belonged to, so if a new user enters the program and belongs to this class, it will be compared to people of this class through a metric and given the report of the person that is closer to them.

The attributes of the user that form the metric are different for each class, and can be things like gender, age, medical specialty, etc.

4.3 Experiment Design

To test this approach to seed generation we have created an

To design the experiment we have used anonymized historical grades data from UCM's Computer Science.

Our dataset contains data about anonymized global degree grades from the nineties, containing the year of graduation and the gender among other variables.

This will be our domain.

This dataset is to be viewed from three different perspectives :

1. Students wishing to know how their grade stands among their peers,
2. Teachers who want to know how the year they've teached has faired compared to the others,
3. The figure of a gender delegate who wishes to know if the grade distributions are different when broken down by gender.

Our objective first is to generate successful seed cases for each class which will then be used as base report generation techniques for each category of user within this domain.

4.4 Experiment results

Chapter 5

CBR : Concept

Abstract: In this chapter we outline and explain the user side of the Case Based Reasoning methodology used to test our tool with humans and to better define its ideal workflow.

We provide both an abstract approach and the decisions taken to make the implementation, while leaving the implementation details for the next chapter.

5.1 CBR Design

Let's first talk about what we mean when we talk about a CBR system.

CBR, or Case-Based Reasoning, is the process of solving new problems based on the solutions of similar problems we may have encountered before.

We can see it as a type of analogy solution making.

It doesn't need an explicit domain model and so it becomes a task of gathering case histories.

We achieve the reduction of the implementation to essentially identifying significant features that define a case, which is in essence a lot easier than creating a model explicitily.

CBR systems basically learn by acquiring new knowledge as cases, which combined with data handling techniques and big data make maintaining large volumes of information easier.

5.2 CBR Process

Case-based reasoning can be formulated for a program to emulate as the process that follows:

1. Retrieve: When facing a new problem, get cases relevant to it from

memory. A case is problem, solution, and, optionally, annotations about how the solution was derived.

2. Reuse: Map the solution from the previous case to the target problem.
3. Revise: Having mapped the previous solution to the target situation, test the new solution in the real world (or a simulation) and, if necessary, revise.
4. Retain: After the solution has been successfully adapted to the target problem, store the resulting experience as a new case in memory.

5.3 Retrieve

Conceptually, we need to get the necessary information about past cases of how they were resolved, that is, mainly what problem it was and how it was solved. We start from the idea that our problem is basically analyzing a new dataset. To do this we provide the frame of domains, which ensures us that what we retrieve was relevant condensed information about the problem in the past, and within that information we have the metrics used to analyze datasets like our current one. Another side of how to solve it is represented by the profile information, which tells us how to solve it for the specific user who is using the program.

5.4 Reuse

The knowledge base is obtained primarily from the enumeration of certain past cases or problems. This is built from the fact that experts (humans) are much better at recalling previous experiences and problems than at creating systems of rules.

As new problems are fed to our expert system (containing the knowledge or memory of previous experiences) to which no past problem can match exactly, the system is capable of reasoning from more general similarities to come up with an answer.

This tries to imitate the generalization capability of humans.

To map the solution from the previous cases to the current problem what we do is run an analysis on the current dataset, and then use the metrics contained in the domain information, and then apply the profile information on the analysis to filter the results.

5.5 Revise

When the users are presented with the new report, they have the ability to modify its contents, both graphical and the information they're presented with. This changes are stored in the profile information, so we have access to the new preferences, thus making a better solution from the old one in the spirit of the revise process.

What we're trying to achieve is giving the user as much power as possible, because that is what will make his or her experience better with each time they use the program, which is the whole point behind the implementation of the CBR based system that we've chosen.

5.6 Retain

The system is able to retain new information by being able to make changes to the elements it stores, namely updating the domain objective analysis with the new analysis and updating the subjective profile report changes with the changes introduced by the users.

This is done only once, after the user has made all the changes and exits the program. No interaction is needed on the side of the user to do this, it is done by default so as to make the whole experience as streamlined as possible.

The memory system present in our system grows and changes by each time we present it with a new case. An important aspect of this memory-based process of reasoning is closely related to automatic learning: our system should be able to remember the problems that it has been presented with and to use past information to solve new challenges.

This is intended to complete our modeling of the human behaviour of CBR, and represent the final step of our CBR system.

Chapter 6

CBR : Implementation

Abstract: In this chapter we outline the technical implementations of the CBR side of our program.

6.1 Information Storage

Because the information we're storing is still low and the metrics are different for each domain the only way to consistenly store the information is through a non-relational database. To provide the information to the experiment and to run this as a proof of concept we can simplify it by storing information in the form of JSON structures, which will be stored as files in a directory accessible by our program. The relevant information for the use case at hand will be retrieved by our program, stored as a run time variable, modified when needed and then stored back to the JSON overwriting the previous structure. If we had a non relational database we could simply update the database instead.

6.2 Retrieve

If we're doing a full non-relational database implementation it's probably better to develop a domainID, because that would allow us to have different domains with the same column names. For the proof of concept though that is not an issue because we've made sure this hasn't happened with the datasets used for the testing.

Retrieving this information is as easy as reading the JSON file and loading into a dictionary type in Python. We know which information to retrieve because each json contains an ID variable with the column list of the datasets from the domain. We do this with a file buffer the standard way. Once it has been loaded the information is handled by two different classes.

6.3 Reuse

The objective information about the domain is loaded and handled by the class that handles the objective information, which will then be passed to the The profile or subjective information about the domain and user is handled by the module that handles the profiles, which will then pass it along to the module that generates the report. This allows us to reuse the previous experiences in the process of generating the report module.

6.4 Revise

There are two ways on which we can revise the information. If the users from the start say what they want then the system associates what they want as if changes have been made to the report that was going to be presented to them. In any case, when the users are provided with the report they can provide feedback through a visual interface by selecting which information is shown and shouldn't be shown, what information should be shown but is not the report, and change the colors/fonts of the report. This allows us a thorough revise step, in which the user has full agency over the results generated by the program and is able to change all the choices that our system has made. This is incredibly important because it makes our system learn a lot from each interaction.

6.5 Retain

Once these changes have been made the objective information is updated and written back to disk by the same module that retrieved it through an store method. The same method is present in the module responsible for handling the profile information, which will update the choices the user has made. This is a crucial step because it is what allows us to retrieve these changes consistenly if this user or a similar one wants to use the system in the future. In the end what we've achieved is a system that grows better with each interaction, providing the users with better and easier use experiences every time they use it.