Master in Artificial Intelligence

An Intelligent Decision Support System for Machine Learning Algorithms Recommendation

Andrei Mihai

advised by
Dr. Miquel Sànchez-Marrè,
Department of Computer Science, UPC

facultat d'informàtica de barcelona (fib)
facultat de matemàtiques (ub)
escola tècnica superior d'enginyeria (urv)

universitat politècnica de catalunya (upc) –
BarcelonaTech
universitat de barcelona (ub)
universitat rovira i virgili (urv)

January/February 2017

Abstract

Machine learning is a very central topic in Artificial Intelligence and even computer science in general. Nowadays, its use in Big Data problems is quite well known. However, while the big data, and machine learning problems in general, are quite varied and in needing of different kinds of solutions, there are as well many different methods in machine learning that can be used. In this work, we propose an application that might help deciding on which machine learning methods a user needs for a specified problem.

The application is an Intelligent Decision Support System for Machine Learning Algorithm Recommendation for which we present the design, which is centered around the combined use of the Case-Based Reasoning and Rule-Based Reasoning, for the recommending process, while also trying to make the system easy to use and manage. We present a prototype of such a system, and the implementation details of the two recommender algorithms. The preliminary testing of the prototype shows it to be a promising tool.

Contents

1 Introduction

In this chapter we present the problem that we tackle in this thesis, what are our motivations behind this decision and what are the issues in solving this problem.

1.1 Problem to be solved

Nowadays, one of the main topics in Artificial intelligence is Machine Learning. It is used in many fields, more prominently in the increasing field of Big Data. Machine learning for big data is used in many commercial sectors such as retail, manufacturing, travel, financial services or energy and utilities [15]. While there are many and varied kinds of problems requiring different kinds of big data and machine learning solutions, there are as well many different methods in machine learning that can be used [23].

Among the plethora of methods and algorithms that can be used in machine learning there are of course categories and types of methods. Some types of algorithms are suitable for some problems, some other type of algorithms, for other problems [23]. Experienced machine learning scientist might have knowledge of this kind, but for new users it might not be obvious where to find such information.

In this work, we propose an application that might help deciding on which machine learning methods a user needs for a specified problem. This application could be used by students, beginners in machine learning or simply users that have no expertise in machine learning.

1.2 Motivation

As a student of Artificial Intelligence I have been given different tasks that needed to be solved with machine learning methods. While in some cases we would receive advises (knowledge from an experienced user) in others we just had to choose the method. However, as stated before, there are many methods of machine learning, and we ended up choosing one that just seemed interesting or simply the first which seemed to work.

Finding information on what is the best algorithm for the problem did not seem to be trivial. Of course, there is no way in which to perfectly predict which is the best algorithm for a given task and no algorithm is best for all tasks [32]. Yet, a choice of what algorithm to use has to be made. Even if many problems using machine learning are solved by trial and error [11], a choice has to be made on where to start. And so the question arose: could a practical Intelligent Decision Support System (IDSS) be created to help make this choice?

And after some research, we found that, between cheat sheets for algorithms and automatic algorithm selection (see chapter 2), there could be potential behind such an Intelligent Decision Support System.

1.3 Issues of the work

The biggest problem of such work is the extremely high number of variations of problem/solution pairs in machine learning (where solution refers to the algorithm used). This high variation makes it hard to pinpoint some clear rules or (maybe even harder) to automatically extract these rules. This leads to vagueness and sparsity and simply recommending the best method for a problem, in an effective way, seems not to be doable with the current technology [21].

And this leads to a second issue, somehow derived from the first one. It is hard to gather the data needed. Even if the system is supposed to recommend approximations, the data needed to fuel such a system has to come from expert knowledge and previously made experiences and studies. Yet, because of the great variation, these are quite sparse, and trying to find clear information on most type of problems and most types of methods in machine learning seems to be very hard.

And the last issue we want to mention is the current growth of machine learning. Machine learning is a very interesting topic, used and usable in many fields (as mentioned in section 1.1), and there is a lot of work concentrated on this field, to make it more powerful, with better methods, applicable on more and more problems and to work better in older existing problems. Simply put, Machine Learning grows and changes at a fast rate nowadays. While this may be good for the field, as it is expanding and getting better, the fast pace of change in it is quite an issue for the type of application discussed in this work. That is, while trying to create an intelligent decision support system to recommend machine learning methods, machine learning is changing. And this leads to the risk of making the intelligent decision support system obsolete faster than it can reach its potential. A way to mitigate, or even overcome, this problem is by making the system easily extensible, so that it can be updated faster to the new technology.

2 State of the art

Nowadays, with so many algorithms and machine learning methods, most of the problems are solved by trial and error, in order to find the best learner, and it may even be considered among the best practices in machine learning [8]. When faced with a problem, a machine learning expert will try different algorithms that in his knowledge could be good for that problem and choose the best one. Notwithstanding, what happens when there is not enough knowledge on the problem or the user is not a machine learning expert?

While it seems that there is no actual research to answer this question, both in our experience and research on "Question and Answer" websites such as Quora[1] or StackExchange[2], what there is to do is to consult an expert that might have some knowledge or simply search the information on the internet trying to find the knowledge you need (where the websites we have mentioned tend to help).

In this context we thought about an Intelligent Decision Support System to help with machine learning, especially for students or beginners.

2.1 Related Work

Of course, trying to find the best machine learning algorithm for a problem is not a new idea, quite the contrary. Even since the earlier days of machine learning, when more and more learning systems started to be developed, it became apparent that these systems usually are only effective on a narrow range of problems, for which they were designed [24]. So efforts were being made to create a more general machine learning algorithm, or an algorithm that could generate the best machine learning algorithm needed [11].

In order to get the best out of machine learning for a certain problem, choosing the best algorithm and hyper-parameters can be viewed as an optimization problem. In [21], Luo explains quite well this approach and summarizes most of the work done in this direction. Luo gave 4 main functionalities that an automated machine learning system can have: efficiently handle big data, handle a wide range of algorithms, handle various types of hyper-parameters, handle any number of hyper-parameter value combinations. Of all the algorithms Luo presented, none of them were able to have all these functionalities. While some of these systems can be used in some domains, there is still work to be done before they can be used efficiently in practice, especially for big data (in which [21] had interest).

The use in practice of such systems can also enable non experts to use machine learning easily [9]. There are tools that were created for this purpose, such as Weka, RapidMiner or Scikit Learn. These kinds of tools were designed to give easy access to machine learning methods, even for non computer scientists. However, only access to a lot of machine learning methods without the knowledge how to use it only solves a part of the problem. That's why

systems such as Hyperopt-Sklearn [9] and Auto-WEKA [30] were researched into and developed. Nevertheless, although there are reported results using such systems, they are still mostly theoretical without the possibility to be actually used in practice yet [11, 9].

Another related research field is that of *meta-learning*. Meta-learning, in computer science, is related to understanding how the learning process is working, and developing automation of these processes [11]. In more practical terms, some system is called to have used meta-learning when learning on the meta-data of base learning algorithms is employed. Meta-learning is quite a wide field, with various type of applications. According to Rostislav Striz [29] the applications of meta-learning include:

- selecting and recommending machine learning algorithms
- combine base-level machine learning systems
- control the learning process and bias management
- transfer meta-knowledge across domains

It is quite clear that meta-learning is a wide domain, but it includes the issue treated in this paper "recommending machine learning algorithms". in [11] quite a good synthesis of algorithms used to select or recommend machine learning methods using meta-learning has been done.

Some of the machine learning algorithms recommenders presented are MiningMart [34], the Data Mining Advisor [10] or Metala [12]. MiningMart is a method that is focused on the data preprocessing rather than on data mining (or learning) algorithm selection [28]. The Data Mining Advisor was an "early result" of the meta-learning research in data mining, although it had more of a limited set of base-level algorithms (for example it did not include SVM) [10]. Metala is a meta-learning method that is focused on web-based data mining. It is more of a framework rather than a system in itself [12]. Although these methods had some early success [11], they are all outdated [28].

However, in [11] there is mentioned yet another meta-learning system that is a machine learning algorithm recommender: Intelligent Discovery Assistant (IDA). Actually, the IDA is a type of technology that uses meta-learning to aid the process of knowledge discovery, and many different IDAs have been developed [11, 28].

In [28] there is quite a comprehensive comparison between many different IDAs. From these IDAs many have different scopes and applications. Still, in the paper it is presented which IDAs are up to date and which one are outdated. Most of the up to date IDAs are the so-called "Workflow Composition Environments". These systems help the user to more easily create a work flow for the knowledge discovery or data mining task he/she needs, but, in general,

do not automatically propose algorithms to use or hyper-parameters. As they help and guide the user through different methods [28] introduced them in the comparison as IDAs. Example of these kind of systems are RapidMiner or Weka, both of which we have already mentioned in a previous paragraph.

Yet, these Workflow Composition Environments are powerful tools, and the need for an actual Intelligent Discovery Assistant that can automatically propose algorithms and hyper-parameters became obvious. Inside the e-LICO European Project there is a research group that develops and IDA that is integrated into RapidMiner [19]. This Intelligent Discovery Assistant can automatically generate a workflow for the data mining or knowledge discovery task, including data preprocessing and choosing the highest ranked learning algorithm. The inputs of the IDA are the data that need processing and the goal (problem to be solved e.g. "predictive modeling"). As a methods recommendation tool, this IDA's drawbacks might be being a RapidMiner extensions (and thus it can only be used by RapidMiner users) and that it needs the whole training data to make the recommendations (which may be cumbersome in data mining on big data).

2.2 Tools and technologies used

While we have presented the related work done in the previous section we will present now the tools and technologies we have used to complete this project.

Intelligent Decision Support System. Generally, a Decision Support System is a software designed to help decision-makers solve decision problems in practice, using specialized data and models. The support of computational technologies help in dealing with complex, uncertain and unstructured decision problems [16]. An *Intelligent* Decision Support System is an Decision Support System that involves extensive use of Artificial Intelligence techniques, such as Case-Based Reasoning, cluster analysis or classification, to help identify rules for the IDSS [16].

IDSSs are widespread in today's informational trend. One example would be that, with the large scale use of the internet different systems were developed to help consumers navigate, most obvious ones being recommender systems (which are a class of IDSS [16]) that can be seen, for example, in most e-commerce websites.

The problem of choosing the best machine learning algorithm for a given problem can be seen as an optimization problem, where there we need to find the algorithm and hyper-parameters that minimize some error function defined for the problem [21]. However, it can also be seen as a decision problem, where the user of machine learning (having a problem to be solved by machine learning) is a decision maker who has to decide the best use of machine learning methods in order to have a better solution for the problem. We have decided

to see the problem as a decision problem and to design an Intelligent Decision Support System to help users with this decision.

Expert Systems An Expert System is an software system that has the ability to imitate the capabilities of reasoning, explaining and managing a knowledge base of a human expert [14]. Usually Expert System are good at providing a way of dealing with expert knowledge that is represented as rules [33]. Rules and fact bases are a relatively good representation for domain knowledge that is informal and badly structured [33]. Even tough expert systems are hard to develop, in a conventional way, nowadays there are hybrid systems appearing that combine expert systems with other methods [27]. More recently, expert systems in combinations with other methods, have been used in medicine, finance, education, fault diagnosis or even industry [27].

The expert knowledge on machine learning methods is quite informal and unstructured. There is no clear information on which methods are better and which are not. Yet there are some indicators, such as experts' answers on question and answer sites or a cheat sheet for a specific machine learning framework (e.g. Scikit Learn or Azure). Although knowledge acquisition is one of the biggest problems of expert systems development [18], the best choice of representing the scarce knowledge that can be extracted from the internet on this topic seems to be rules and expert systems are good at working with rules. Hence the proposal we have made is to use expert systems.

Case-Based Reasoning Expert systems should be supplemented with another method in order to improve the results, and our proposal was Case-Based Reasoning.

Case-Based Reasoning can be viewed as a paradigm for Artificial Intelligence research as well as as a methodology for developing practical systems [3]. CBR is a method or process of solving problems that involves retaining some experiences (or cases) in a case base, retrieving the most appropriate cases when a new problem arises, revise then reuse those cases to solve the new problem then retain the experience of problem and solution as a new case. CBR can be used for problem solving as well as interpreting the state of the world [3].

CBR is a complex field that has roots in many different research fields such as mathematics, cognitive sciences or machine learning [25] and unlike some other methods derived from artificial intelligence research, there were CBR applications successfully used in a customer used environment even at the first appearances of the method [6]. CBR is used commercially as well as in research in fields such as engineering, help-desk and customer support, e-commerce and so on [6]. There are even medical applications that are developed upon the Case-Based Reasoning methodology such as diagnosis and decision support systems [13]. And then, there are CBR recommender systems. CBR is a key component of the content-based recommender systems, which is one of the

main types of recommender systems [4]. There are numerous applications that successfully use recommender system based on CBR [4].

Actually creating a way to determine objectively what machine learning algorithm is the best only using metadata, independent of previous problems is hard [11, 21]. The expert knowledge is rather scarce and convoluted, and only an expert system might not give good enough results. Taking into account that there are quite a few papers on how different machine learning algorithms perform on different data, using Case-Based Reasoning to support the decision of choosing machine learning algorithms seems a reasonable idea.

Python Python is a general purpose, strongly and dynamically typed, language. It can be defined as "Object Oriented Scripting Language" [22]. Python is a widely known and widely used language, at the moment of writing in October 2016, ranking number 5 in Tiobe index and second in PYPL index of popularity of programming languages. It is used for diverse systems; it can be used for small scripting purposes or it can be used in big commercial applications in which is most notably known for its use in applications such as Google search, YouTube or BitTorrent [22].

Of course there are other widely used languages with which one could create a fully functioning computational system, but there are several reasons for which we choose Python:

- First and most importantly, I have more experience with Python than with other languages. During my education, Python has been one of the most used languages (along with Java), and I have also implemented my bachelor's thesis project in Python (as well as Python being my first choice in my side projects)
- Python is designed to be readable, and thus easy to manage and maintain
- There are many libraries created for Python and there is a very active community and thus there is a lot of support for most things one would like to implement in this language
- There are many powerful machine learning libraries for Python such as Tenserflow, Scikit-learn, Theano or Pattern.

3 Machine Learning Recommender Proposal

In this chapter we will present the system we propose to solve the issues we presented in the previous chapters. We will start by clearly explaining how the system is envisioned to be used and its use cases. Then we explain what kind of data is needed and finally a high-level architectural design proposal for the system.

3.1 Overview

In the previous section we mentioned several times what is the problem that we want to solve. Now is the time to explain how we think it may be possible to solve it. In short, our proposal is a system that, given a task (which can be solved using machine learning), recommends at least a starting point of a solution (a machine learning algorithm).

Now, what we think of as a task that can be solved by using a machine learning algorithm is in general a knowledge discovery and data mining task. That is, there is a dataset on which a process needs to be performed in order to meet a goal, and the task is to apply the right process on the dataset. Our system is designed to ease this task. Thus, based on some information about the dataset and about the goal, it should give recommendations on how to design or implement the knowledge discovery or data mining process.

Therefore, the input of the system is the information about the data and the goal. Information about the data means meta-data of the database which includes more neutral information such as the number of samples of the database, the number of features, if the data is expected to be noisy but also information about the knowledge in the data, such as if the data are images, audio, multivariate and so on. Information about the goal refers to the reason for which a model needs to be extracted from the data. In general, that may be classification, prediction, clustering and so on. However, if the data is of a certain type, the goal may be more specific, such as object recognition from images or music extraction from audio files, as there might be some algorithms that are good for some specific problems that appear often, (e.g. face recognition from images). In addition, there might be some meta-data about the dataset that is specific to a certain goal, for example, for binary classification problems knowledge of the proportion of the classes is relevant (i.e. if there is class balance or not). Also, depending on the goal, different meta-data is needed, for example, if the number of clusters is known, in case of clustering.

So the the meta-data needed of the problem can vary depending on the type of problem, so the system has to accommodate this variance in input. The answer to such a specific question can be hard to find, but the system is not designed to give an exact answer. It was designed to give a "nudge" in the right direction. A recommendation on where to start or what might work. Specifically, the output of the system is not one exact configuration,

but multiple algorithms that should be viewed as options of what might work. In addition, the system outputs other data mining steps that might be relevant. For example, feature scaling before feeding the data to the algorithms, if some of the recommended algorithms are not scale invariant. In addition, the system might output other information that might be relevant to the user, such as different resources. For instance, if it recommends Support Vector Machines with a non-linear kernel, it might also give a resource where different types of kernels are explained, so the user can more easily implement the recommended algorithm.

In short, the proposed system takes into account meta-data about the problem, which can vary greatly, depending on the problem, and returns multiple recommended algorithms, possible with some meta-data (hyperparameters) configurations of those algorithms, possible recommendations for other steps of the data mining process, and possibly other information that might be relevant to the user.

As technologies to power the proposed system were chosen Rule-Based Reasoning and Case-Based reasoning. A very important characteristic in both methods is that they can easily handle input data with varying attributes. Unlike other types of methods, inference on rules and cases' retrieval can be done with highly different input without effort, if enough rules and cases are provided.

Considering that most potential users, in the absence of a system such as the one proposed here, used "rules of thumb" found on the internet or given by experts in order to start implementing the data-mining process, we thought that a rule-based system would be a good fit for our system. Even if such rules are not perfect, they are good enough to guide users. Gathering many such a rules together in one rule-based system can give better guidance than the sparse rules of thumb one might found, and this guidance is much easier to reach and use. One of the major drawbacks in rule-based systems is that they become unmanageable as the number of rules increases greatly, both in terms of computational time and rule base management. On the other hand, our system does not need to give the perfect answer, so that an extensive set of rules to cover all possibilities is not needed. Thus, some rules for more general recommendations and maybe for some specific prevalent problems, should be enough for a good-enough guiding. Considering this, we don't expect the size of the rule base to become unmanageable.

Another drawback of a rule-system is the fact that they are not flexible, they cannot adapt to an changing environment. And the world of machine learning and data-mining is quite dynamic, especially nowadays when Artificial Intelligence is being developed rapidly. While there might be some rules of thumb that pass the test of time, a better guidance is given if the rules are adapted to the new algorithms and variations, and to different problems. Thus, in order for such a system to work, there is need of constant update of the rule

base by experts.

Although, this is exactly one of the strengths of the case-based reasoner. We decided to add a case-based reasoner because we thought that recommendations from general rules might not be enough. The case-based reasoner is thought to give recommendations from real cases, i.e. from other experiences. Thus, the system would give guidance based on rules of thumb and previous experience. We observed that the two systems are quite complementary. While there is constant need of experts to manage the case base, the systems encourages users to add their final solution to the case base, so that, if the system is used, the case base is always up to date with the new types of problems and solutions. On the other hand, while the rule base does not need an excessive amount of rules to function properly, the case base tends to grow constantly, potentially getting to big to effectively use it for reasoning, or to manage it. This could be overcome by having experts that manage the case base or by having an automated process that removes less useful cases.

As this system is to be used mainly by non-experts, ease of use was also taken into account. The meta-data of the problem is given to the system using an information extraction approach. The user is given at first some general questions about the problem, such as the goal or the number of samples, and more specific questions appear as the user gives answer. This functionality is also powered by a rule-based systems. Moreover, there is the need to constantly manage the rule base and case base, so the system provides tools to ease the managing of rules and cases.

3.2 High-level Design

In this section we will present the high-level design of our proposed system. That is, how we envision the system should work and formally present it.

We have clearly stated in the first two sections who are the users of our proposed system. On top of that, there should be administrators and/or researchers that manage the rule base and case base so that the recommendations are up to date. The different ways that our system can be used can be formally described by a use-case diagram.

Figure 1 shows this system's use-case diagram. The two main actors are the user and the administrator/researcher, who has expert knowledge about machine learning and different algorithms. The usual user mainly needs recommendation about a specific problem, which needs that the user enters meta-data about the problem. Also, the user needs to be able (and should be encouraged to) to insert cases in the case base, that is, insert the final solution to his/her problem. The expert (administrator or researcher) needs to be able to manage the case base, that is, to add cases, change cases (if they are outdated or incorrect and so on) and delete cases (if they are not needed anymore or if they are redundant). I also needs to be able to manage the rule base and fact

Figure 1: Use Case Diagram.

base in order to keep the rules up to date and also keep the information extractor up to date with the modern problems (new type of meta-data needed) and algorithms.

In order to accommodate these needs, the system needs clearly separated sub-systems that can be accessed and controlled through a Graphical User Interface (GUI). The system has a rule-based sub-system that includes an information extractor and the recommender. The system also has a separate case-based reasoner that can give recommendation based on a given problem. Also, the system has a Graphical User Interface with which the actors can interact with the system. In order to combine the functionalities of these sub systems it needs a controller that links the inputs from the Graphical User Interface to the right system (rule-based reasoner or case-based reasoner) can get the answers and give them to the GUI in a way they can be displayed

properly. The general flow of the system can be seen in Figure 2.

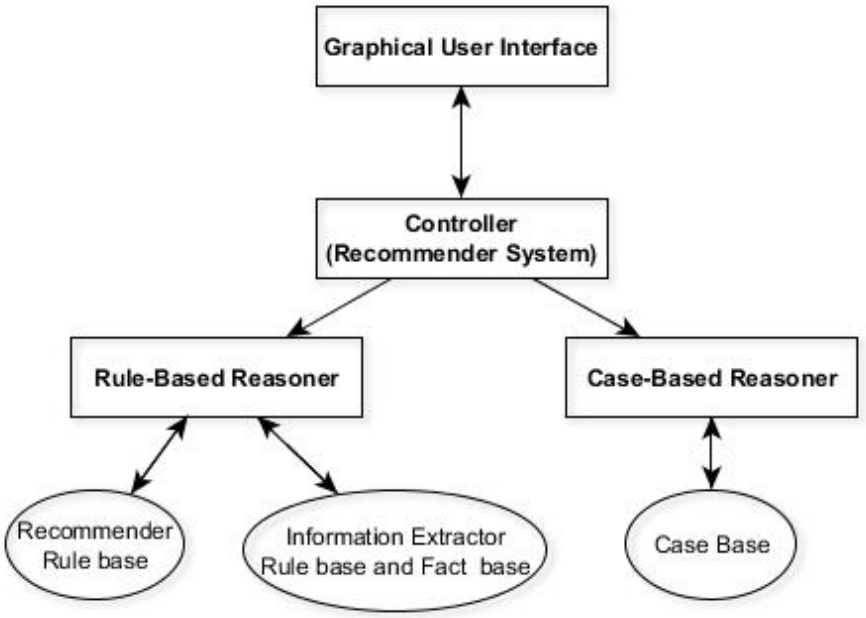

Figure 2: General flow of the system

Now, we will go a bit into details on how the separate systems are designed. We will not go into details of the Graphical User Interface, as it really dependent on the implementation and we will present more when we talk about the system prototype we developed. The three main systems are the case-based reasoner, the rule-based reasoner and the controller. The controller is actually coordinating the whole recommending process, so when presenting the controller design, we actually present the design of the recommendation process.

One of the subsystems is the *rule-based reasoner*. The rule-based reasoner has two roles: information extraction and recommendation. In Figure 3 the general design of the rule-based reasoner is presented. There are three main parts of the rule-based reasoner: the information extraction workflow, the rule-based recommender workflow and the rule base and fact base management. Simply put, the management of the rule base and fact base are general CRUD operations: create new rules and facts, update some rules and facts with new information or correct them and delete obsolete or wrong rules and facts. There are two rule bases and one fact base. The rule-based recommender uses one rule base, containing rules about what recommendation to give in what situation. There is no static fact base for the rule-based recommender, as all the facts are given by the user, through the Graphical User Interface. There is one rule base for the information extractor, containing rules expressing what

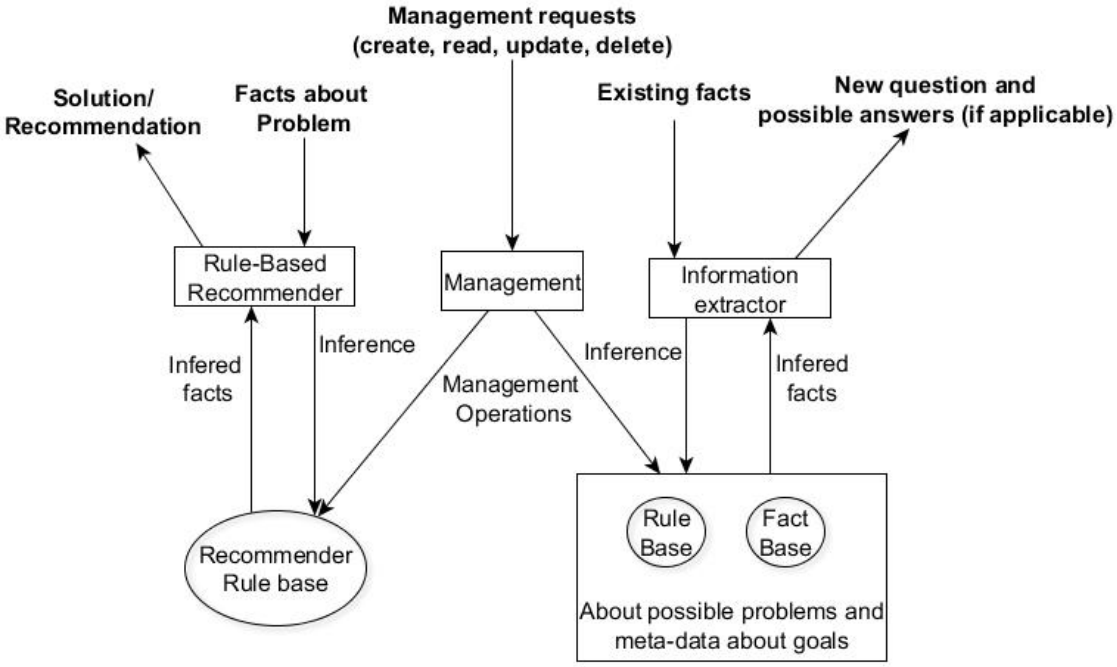

Figure 3: The rule-based subsystem design.

questions to ask the user in what situation. There is also a static fact base for the information extractor containing information such as what are the possible answers for different questions.

There needs to be a subsystem that can manage these information bases, which is represented by the "Management" square in the figure. The "Management" subsystem gets requests to modify the information bases from the user (through the Graphical User Interface and the controller) and it performs management operations on the bases, also assuring their persistence.

The *Rule-Based Recommender* subsystem receives facts about the problem, not necessarily in a representation that the inference engine can use, so some conversion might be necessary. Then it uses the inference engine to infer facts from the given data and the existing rules about recommendation. Finally, the rule-based recommender assembles the inferred facts in a solution usable by the controller (which will pass it to the Graphical User Interface which will display it to the user).

Finally, there is the *Information Extractor* subsystem that receives facts about the problem, and similarly to the rules recommender, might need to convert them in order to become usable by the inference engine. Then it uses these facts and the facts in the fact base to infer what other questions need to be displayed to the user, based on the existing rules.

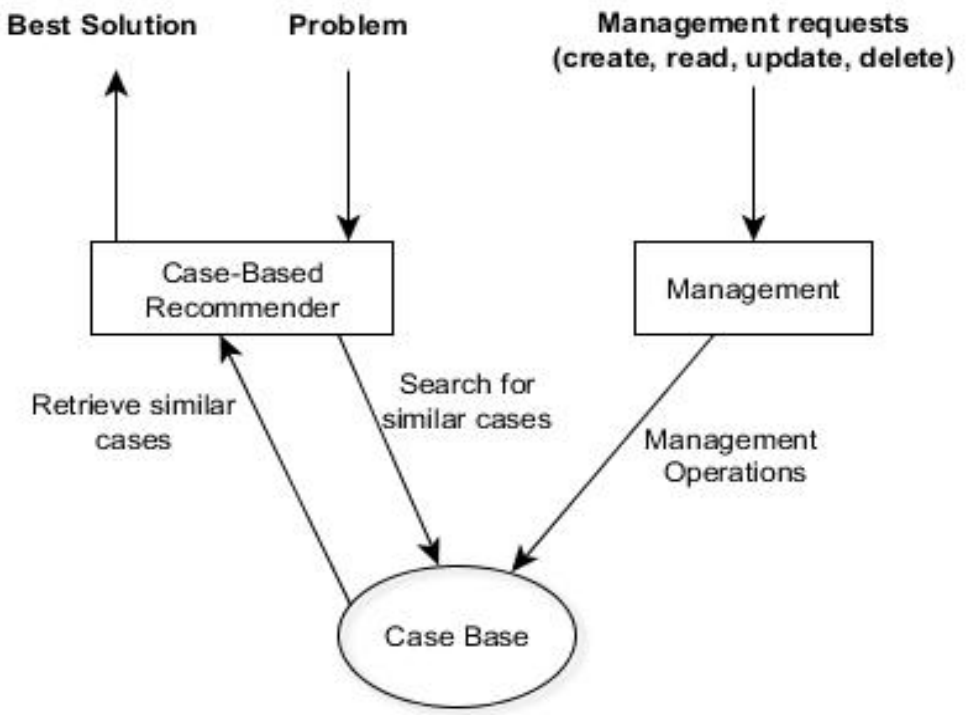

Figure 4: The case-based reasoner subsystem design.

The second part of our proposed system is the *Case-Based Reasoner*. In figure 4 the general design of the case-based reasoner subsystem can be viewed. The Case-Based Reasoning is visibly simpler than the Rule-Based Reasoning, since the Case-Based Reasoning only has one main goal: giving recommendations (as opposed to the Rule-Based Reasoning, which also is used for information extraction). There is one case base, that the case-based recommender uses. Each case contains a pair of problem-solution, that were added by the system's administrators, researcher or experts, or (and this should be true for most cases) added by the users themselves, once they successfully finished a data-mining task, so their solution can be used in the future by another user.

As mentioned, this case base needs to be able to be managed by administrators, researchers or experts, so there is need of a "Management" subsystem, as in the case of the rule-based recommender above. Very similarly to the aforementioned, this management subsystem receives CRUD management requests from the user, through the GUI and the controller, and modifies accordingly the case base, assuring that the modifications are persistent.

The case-based recommender receives data about the problem from the user and, like in the previous cases, it might need to convert the received data to be properly used by the reasoner, depending on the implementation. Then it uses the given data to search for similar cases. The similarity is computed

comparing the problem part of the case, with the given problem. There might be a need to completely ignore some existent cases in the case base, which may look similar but are of no actual use for the given problem. For example, even if the dataset is the same, if the goal is dimensionality reduction instead of classification, then the case should be ignored, as algorithms for classification may perform poorly on dimensionality reduction tasks, even if it is the same dataset.

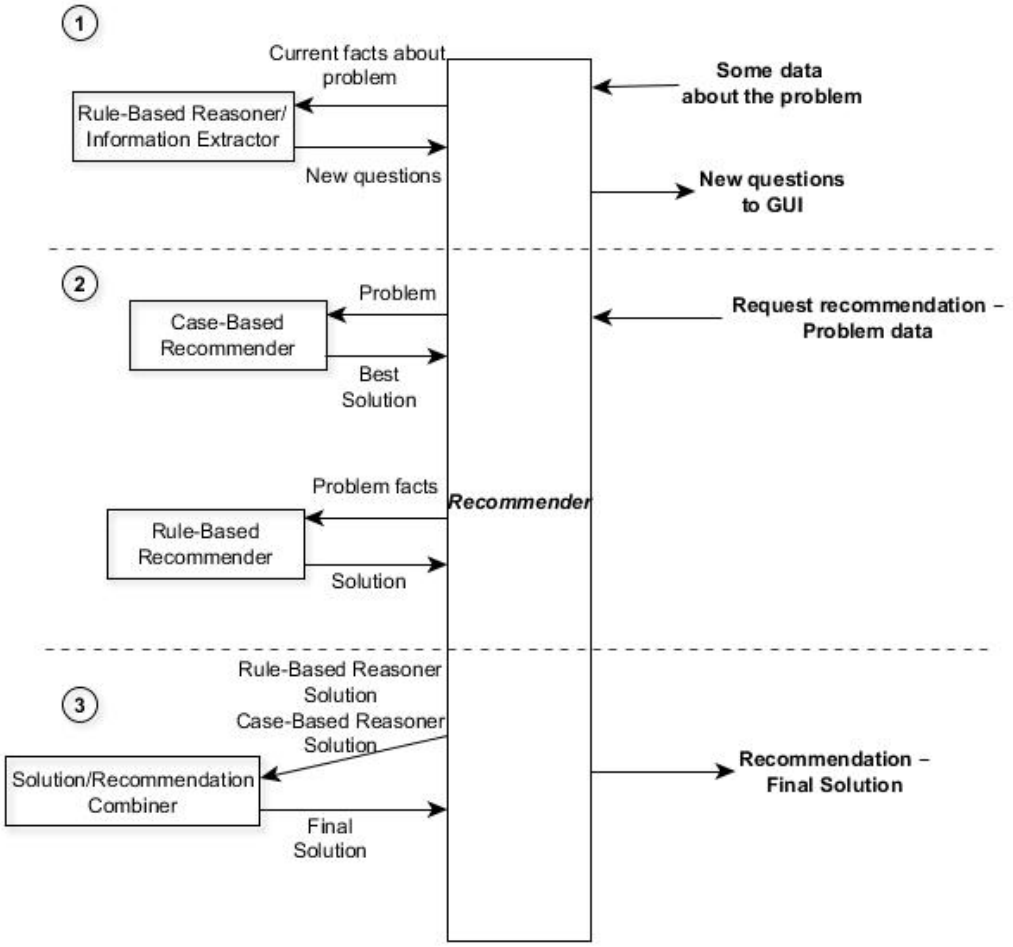

Figure 5: The general recommendation flow. The controller part of the system takes the role of "the" recommender.

And finally, there is the *Controller* part of the system, that makes the link with all of the other parts. We viewed the controller as playing the role of "the" recommender, as in, being the overall recommender that the user uses, encompassing in its structure the information extractor, the rule-based recommender and the case-based recommender. The controller also has the role

to pass on the management information to the respective subsystem, but this purpose is more of a secondary one, the primary one being that of coordinating the recommendation process and giving for display to the Graphical User Interface the final solution i.e. the recommendation requested by the user. In Figure 5, the flow of a complete recommendation as well as the controller's role as "the" recommender can be seen.

Now, we will focus only on the recommender part of the controller, as the management part is less interesting and quite trivial (as the controller only passes the request from the GUI to the respective subsystem). In our view of the system, *the recommendation is performed in three steps*. First, the information about the problem is extracted, then the solutions are requested from the two recommenders and finally these solutions are combined to form a final solution to give to the user.

- In the **first step**, the user sends with the help of the Graphical User Interface some data about the problem. The Controller than passes these data to the information extractor, that infers what other questions might be asked about the problem to the user. Then the questions are passed to the Graphical User Interface to be displayed to the user. Based on the new answers from the user, this process might be repeated, until enough data is gathered from the user about the problem.

- The **second phase** of the recommendation process also starts at the signal of the user, as the user has to finish uploading the data of his/her problem into the system. Then the user makes a request for a recommendation. When the controller receives the recommendation, it takes the inserted problem data and passes them to the case-based reasoner, requesting for a recommendation, and to the rule-based reasoner, requesting for a recommendation. The order of the request is not important, the recommenders can be requested in any order. The recommenders can even work in parallel if the implementation enables it and it seems necessary, or that it improves the recommendation process. In any case, the next phase starts when the controller receives both of the solutions from the two recommenders.

- When the controller has received both recommendations, the **third step** starts. While solutions have already been given, the controller can give only one solution to the GUI for displaying to the user. Therefore, for this, the two solutions given need to be merged. Thus, we envisioned another subsystem that we named "Solution/Recommendation Combiner". This subsystem takes two solutions and outputs only one, combining the recommendations given in the two solutions. We did not provide a general design for the Recommendations Combiner, as it is quite dependent on the representation of the solution in a particular implementation of the system.

3.3 About Intelligent Algorithms used

In this section we will focus more on the role of the rule-based system and the case-based system in our proposed methodology. Also, even if not technically considered an intelligent method, we will shortly study more closely the combination of the output of the Rule-Based Reasoning system and the Case-Based Reasoning system.

Rule-Based System The first intelligent system used in our system is the rule-based system that supports recommendation and information extraction. We envisioned this rule-based system to be something akin to an expert system, mimicking the questions that an expert may ask about a given problem (the information extractor) and the rules of thumb he may give in order to help with the task (the recommender).

There are two main components that are a part of a rule-based system: the inference engine (both forward and backward) and the knowledge given in form of rules and facts. The inference engine is a general algorithm that given a set of rules and facts, can infer new facts following some kind of logic formalism (e.g. first-order logic). Of course, the rules and facts are supposedly following the same logic formalism. The inference engine is independent of the problem and of the rules and facts it is needed to be used on. The only link between the inference engine and the actual problem it is used to solve is the logic formalism. That is, the data needs to be able to be represented in the formalism that the engine works with. There are much more problems to be solved with a rule-based system than there are logic formalisms, thus there can be one problem to create an inference engine for a logic formalism, and a completely separate problem to represent a specific problem in a logic formalism and use the inference engine made for that formalism to solve it. That is why the second main component when using a rule-based system is the representation of your problem as rules and facts in the logic formalism of choice, in a way that using an inference engine, the problem can be solved.

While an already existent inference engine can be used in the system, the problem that the rule engine poses is the translation of the information the system uses into a logical formalism. So the question is, how to represent the problem meta-data as facts in this logical formalism, how to represent the solution or the recommendation as facts, and how to use the particular logical formalism to represent the rules that lead from one to another (from meta-data to recommendation). Also, the information extraction uses a rule-based system. The same inference engine, or a different one can be used to power the information extractor. In any case, the question of how to represent questions and possible answers remains.

It is very important to note, that the way to represent the facts and rules for the information extractor must be linked to the representation of facts and rules for the recommender. Because, the information extractor may give possible answers to a question it sends to be asked to the user, and the answer the user

gives will be used as meta-data fact about the problem by the recommender. Thus there needs to be a kind of understanding between the facts generated by the information extractor and the facts that the recommender can use. This understanding needs to be implemented (at least in part) at the formalization level, so that the facts are represented in the same way, and the expert entering the facts needs to make sure that the facts from the information extractor are understood by the recommender.

Case-Based Reasoning Case-Based Reasoning is the second intelligent method used in our proposed system. In the traditional CBR system there are four basic steps: retrieve, reuse, revise and retain [26].

In the retrieving phase we need to find the cases that had the most similar problem to the one given by the user. This is the most important step in our system's Case-Based Reasoning, as the better match the reasoner finds for the problem, the more will the solution given help the user. It is critical to develop a good similarity function between problems. While this is dependent on the implementation, there are some higher level decisions that need to be made, based on knowledge of data-mining problems and algorithms. For example, as mentioned before, maybe some cases should not be eligible based on the goal (if the problem has the goal of dimensionality reduction and the problem in the case had the goal of classification, then maybe the solution of the case is not applicable to the problem). Or, maybe some meta-data needs to be given different weights, for example, giving more weight to the data type (image data, sound data etc.) than to the number of samples, although both might be more important than the level of noise in the data. These decisions are made having knowledge about different problems and machine learning algorithms. While there is no perfect similarity function, expert knowledge can help. What is even more important is that the similarity function's efficacy my change in time as different types of problems and algorithms arise that are maybe affected by different data in the problem. Even if we did not design our system for it now, it may be a future development the addition of a Graphical User Interface sustained modification of the similarity function, so that the experts that manage the case base, might also manage the similarity function, and update it if needed.

The reuse phase in our case, refers to returning a solution to the controller. This can simply be a straightforward action, such as returning the solution of the case with the problem most similar to the given problem, or might be more complex, such as combining the solution of the two or three best cases.

The revise case is not done automatically by the system. Instead it is done manually by the user. The user takes the combined recommendation of the rule-based reasoner and the case-based reasoner and tries to solve its problem, to complete its data-mining task. Then, when they managed to successfully complete the task they had, they will add a revised version of the solution, to the case base; the solution that actually was the best for them. Thus the

retention phase is done manually by the user, after they revise the solution.

As stated in a previous section, the case base needs to be managed, as to remove redundant cases and to update others with more new algorithms. One idea for future improvement might be to implement an adaptable version of Case-Based Reasoning that can do some of the managing work, for example as the one described in [26].

Combining outputs Combining the two solutions is closely tied to the two previously described intelligent methods. The core idea of this Intelligent Decision Support System is that it combines the strengths of the rule-based reasoner and the cased based reasoner, so that it can give a better recommendation (and mentioned in a section above, was the fact that the rule-based reasoner and case-based reasoner are complementary in a sense). Thus, then arises the problem of actually combining their output.

The first challenge is to represent the outputs of both reasoners so that they can be compared, used together and combined. Well, in our case, even more than that, as the rule-based reasoner is powering the information extractor, which makes the user to add more data to the problem. When the user asks for a recommendation, the data gathered is sent to the case-based reasoner to be compared to other problems. So the data gathered dynamically with the help of the information extractor needs to fit the representation that is needed for the similarity method to compare problems. So, when implementing this system that combines the two methods, the fact that both methods are used should be taken into consideration throughout all the implementation, and not only at the solution combination stage. This might reduce to the problem of finding a representation of the problem and solution that can be the same for the rule-based recommender subsystem and the case-based reasoner subsystem.

Thus, at the point of combining the two solutions, there should not be representations issues, as both solutions should be represented the same. There is still the issue of combining solutions left. This is largely dependent on the implementation of the solution, but there might still be issues for some parts of the solution, where there is need to choose. How to choose which solution is better. At these point, again, there is need for expert knowledge, to choose what might be the most useful choice to display to the user.

CBR provides more concrete knowledge (coming from real experiences) while the Rule-Based Reasoner provides more general knowledge (coming from theoretical background). Therefore, a good criteria could be to give preference to the CBR recommendations, in case of contradictory or very different recommendations.

4 Development of the Recommender

In the last chapter we presented our proposal and gave the high level design of the system. In this chapter we will present the implementation of the developed prototype of the system.

4.1 General System Implementation

In this section we will present the general implementation of the system we described in the previous section. We decided to implement the system using Python 3.5, because it is easy to use and has powerful packages and libraries, such as pyDatalog, which we used for the inference engine, and PyQt which we used for the Graphical User Interface. We stated more reasons in chapter 2.

As expected, the prototype has three main components: The Graphical User Interface, the rule-based recommender, and the case-based reasoner. There is also the Controller, that links the three main components. Also there is the need of a data folder, which contains the case base, recommender rules, extractor rules and the extractor facts. Then, there is the need for the representation of the different data the system uses, such as the problem, the solution, an algorithm and so on. These representations were organized in a package named Entities. In Figure 6 a simplified class diagram of the system can be seen that summarizes the structure of the system. In addition to these files and packages there is a main file, which needs to be be run to start the program, and a `functions.py` file that contains some custom functions needed in some parts of the system. Now we will go into more details about each of these parts.

The *Graphical User Interface* is the most important part when thinking about the ease of use of the system. Our proposed system relies on the ease of use, as users need to easily and clearly get recommendations and, more important, easily add their final solution to the case base. If the users are not incentivized, or have a hard time to add their solution of the problem to the case base, then the system loses one of its main strong points: continuous adding of real and revised cases to the case base, thus always keeping it up to date as the users keep using it. Also the administrators and/or researches need to easily change and manage the case base and rule base and fact base. This is done much easier through the Graphical User Interface, and the easier is to manage the information bases the more effective will the managing be, thus making the system better. The main window of the GUI can be seen in figure 7.

The main purpose of the prototype is to show how the different functionalities of the systems would work and to test the rule-based system and the case-based reasoner. The Graphical User Interface was designed to be easy to use, but it is not what the actual usable system would have. Thus, there

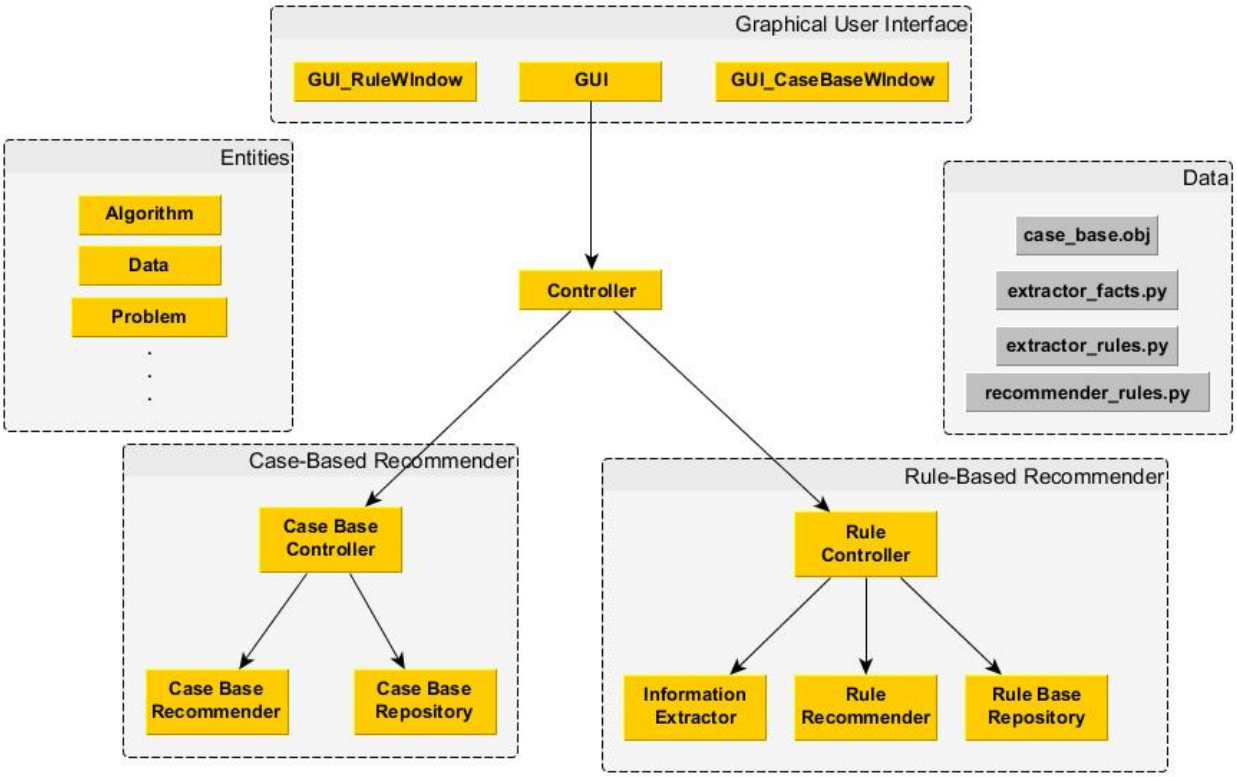

Figure 6: Simplified class diagram.

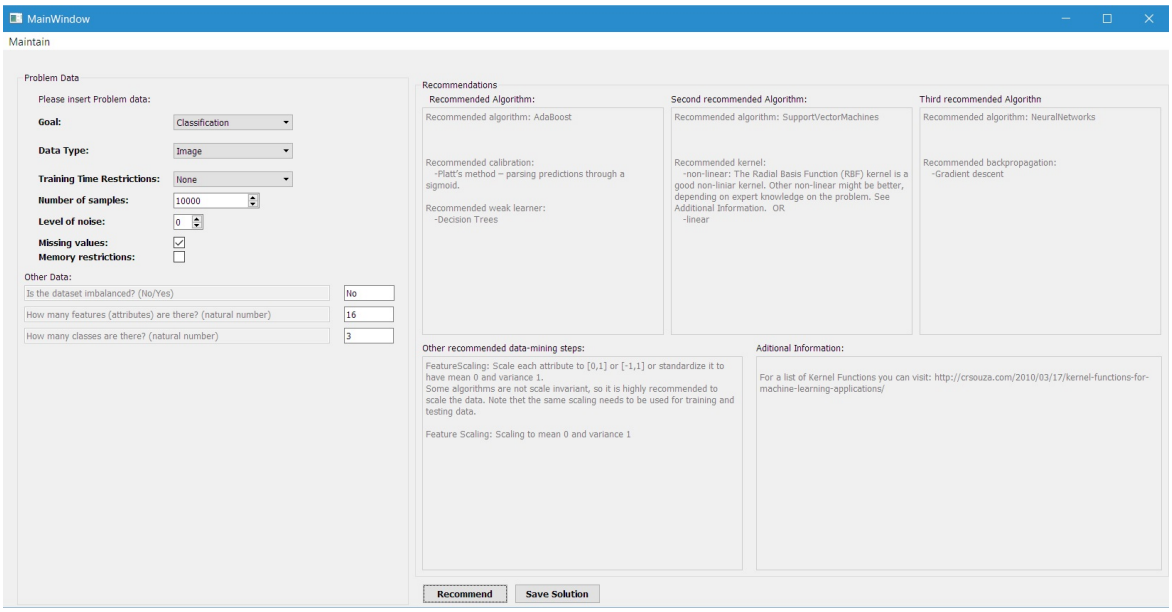

Figure 7: Main window of the GUI.

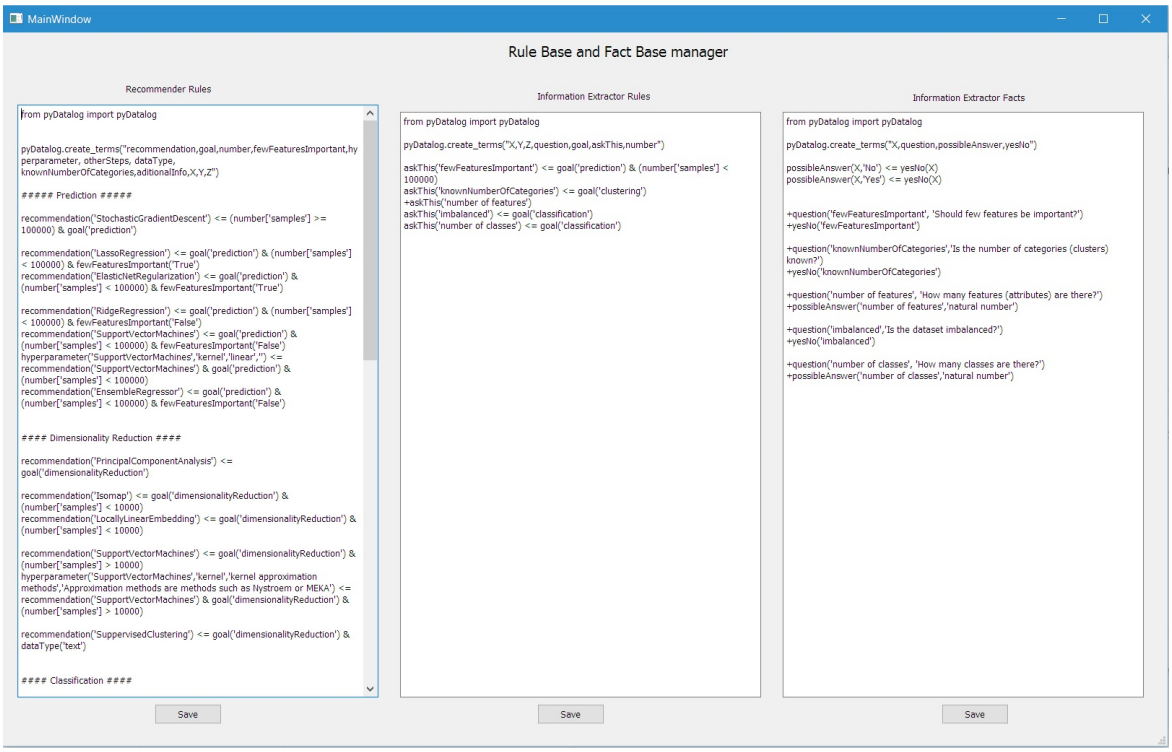

Figure 8: Rule base management window of the GUI

are some functionalities may be unfinished. The rule manager (Figure 8), for example, is for now a simple text editor. The ideal would be if there were some more graphical representations for the rules and the facts, making the managing job just a few clicks here and there and sometimes entering the name and/or details of an algorithm. However, because of lack of time this prototype has a text editor for the rule-based recommender and information extractor data management. Also, the case base manager (Figure 9) uses only a simple list of cases, which is enough for a small number of cases, but not with a large number of cases. Not only that it is slow, but it is somehow hard to manage. For the best management environment, some filters and/or search is needed, sorting and maybe even organizing cases, with a tag method for example.

The rule-based recommender part of the system was implemented using three different classes: `RuleRecommender`, `RuleBaseRepository`, `InformationExtractor` and `RuleController`. There is a need for RuleController, so that the Controller does not need to know about how the rule-based Recommender is organized. The Controller send the requests to the `RuleController`, which in turn sends the requests to the right class. The `RuleBaseRepository` accesses the data files that contain the rules and facts needed for the recommender and the information extractor and performs the read/write operations. The `RuleRecommender` provides a recommendation,

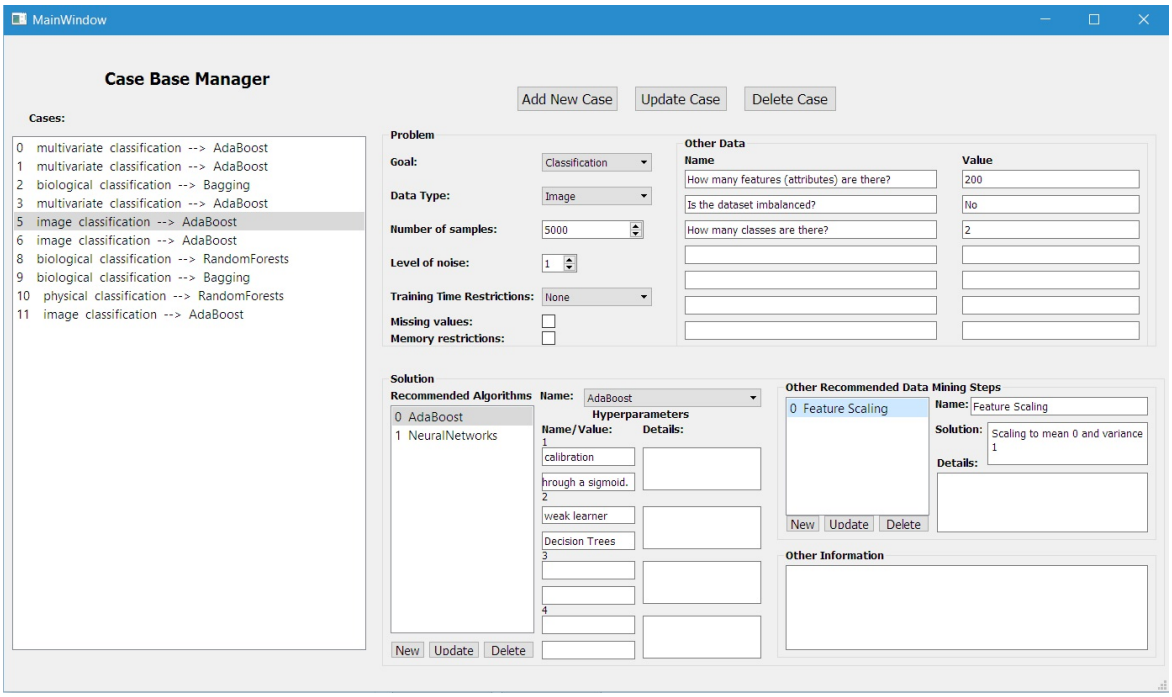

Figure 9: Case base management window of the GUI.

given a problem and the Information Extractor provides an updated version of the problem, that contains new questions, provided a problem. More details about the functioning of the recommender and the information extractor are provided in the next section.

The case-based reasoner was implemented similarly to the rule-based recommender. There is a `CaseBaseController` that performs the communications with the Controller. Beyond mediating the information stream between the Controller and the recommender and the repository, the `CaseBaseController` also keeps the id number for cases. Each case needs an unique id, so there needs to be a way to give newly created Cases an id and to make sure it is unique. Thus, the `CaseBaseController` has an static attribute that takes integer values that serve as identifiers for the cases. Also the `CaseBaseController` creates the actual Case object when the user gives a command to create a new case. Each time a new case is created the static id identifier increases with one and the case is given that id. At the start of the program the Repository searches for the highest id in the case base and the id is initialized with that value.

Then there is the `RuleBaseRepository` that makes the link to the file that contains the case base. The case base is not saved in a database, as that was to much for a first prototype that would have fewer cases. We decided to save the case base in an file by serializing the list of cases from memory. Thus,

the `CaseBaseRepository` also performs serialization and deserialization. These operations are performed with the pickle package provided by Python.

The Controller links the Graphical User Interface with the rest of the system. The Controller also takes the role of the recommender as described in the previous section. We decided to implement the recommender combiner inside the Controller, so that it also takes this role.

The data is organized in four files. One for the case base and the other three for the two rule bases and the fact base. As mentioned before, the case base is saved as a serialization of the list of cases. The file is a .obj file, as this is the convention we use in general for files containing serialized objects, even though the standard for serialization with the pickle package from Python is to save the files with the extension pkl or pickle. The three other files are Python files. This is because the chosen inference engine is the one from pyDatalog, which is a Python version of the Datalog declarative logic programming language. PyDatalog supports the combination of Python code and pyDatalog statements. For that reason, the rules and fact declarations can be done in a Python file. Thus, our rule databases are Python files containing the declarations of the rules in pyDatalog syntax (which uses first order logic formalization and has a syntax similar to Prolog).

In order to be able to implement the system, many elements needed a representation in the object-oriented formalism that Python uses. The main challenges were the representation of the problem and solution. We had to decide what data would these two classes contain, and how will that relate to the user. It was important to design with the user in mind, because it was the user who would insert data and read the recommendations, so that the problem and solution would have to reflect how the user would see them.

We decided that the problem would be split into two main parts:

- first would have some *general data about the problem* such as the goal, the number of samples or the type of the data. Since these attributes would have to be compared with those from other cases (and with those from the rule-based recommender), we decided that it was better to create several Enums for the attributes with a nominal value, such as data type or goal. The problem with this decision is that, when a new type of goal or data type appears and would have to be used in the system, the source coude would have to be changed in order for the Enums to be updated. We thought that, for a future improvement, Enum editing using the Graphical User Interface might be a good addition.
- The second part of the Problem would be the *additional data* the user can insert that is specific to that own problem (e.g. if it is a classification problem, the proportion between classes). We decided to create another

class `Data`, to represent this other data about the problem that might be entered by the user, and the Problem would have a list of these `Data` objects. A `Data` object has a name, which usually is the question that is asked to the user, a value, which is the value given by the user, a list of string representing the possible values, and another string representing an id code. The possible values variable, and the `idCode` variable are needed in order to be able to use the data with the rule-based recommender. Inside the rules formalization, the data will be identified by its id code rather than its name. And, if it would be a nominal attribute, it is important that the values given by the user are matching with the ones inside the rules, and for that reason there are possible values given to the user. Of course, this means that there is a need for coordination between the information extractor and the rule-based recommender, so that the possible values given can be recognized by the rules recommender.

We needed to strictly decide what would a recommendation mean. What should a solution to the user's problem be. As stated in a previous chapter, we viewed the recommendation more as a guideline rather than an exact answer that the user needs to use, and that the recommendation means one or more algorithms, some of their hyperparameters, some other data mining steps and maybe some other information relevant to the user. Thus, we decided that the `Solution` should be represented as a class that has three attributes:

1. *Recommended algorithms*, which is a list of `Algorithm` objects,

2. *Recommended other data mining steps*, which is a list of `DataMiningStep` objects,

3. A string that represents the other relevant information.

An `Algorithm` object has:

1. a name, which is a value from the Enumeration that contains the possible algorithm names,

2. a list of `Hyperparameter` objects.

A `Hyperparameter` object has a name, a value and details, all being strings. `Hyperparameter` would be better represented as a Struct, but there are no Struct structures in Python, thus `Hyperparameter` is a class with public attributes.

A `DataMiningStep` object is similar to a `Hyperparameter` object. It has three attributes: name, solution and details.

And finally, the `main.py` file is used to start up the Graphical User Interface and create the controller. The *functions* file contains three functions, one that transforms a user inserted string to boolean (thus covering cases such as True, true, yes etc.), one that checks if a string is a number, and finally one that prepares two user inserted strings for comparison (thus removing cases and some punctuation).

4.2 Details on Intelligent Methods

In this section we will present in more detail our approach at implementing the needed intelligent methods.

Rule-Based Recommender The first implemented method is the rule-based recommender. As we mentioned before, we have used an already implemented inference engine, more specifically, the one used by the pyDatalog package, which is a Python implementation of the datalog logical programming language. Thus, the challenge was to find a way to represent the problem and solution as facts in order to be able to create facts from the problem data and to create a solution from the inferred facts.

For the *Problem*, we decided to represent the attributes of the problem as a predicate and the value of the attributes as the predicate parameter. So, for example, is a problem would have the goal classification (`problem.goal = Goal.classification`) then in the rules it would be represented as a predicate goal with the parameter classification (`goal('classification')`). A special case are the attributes that are not nominal, that is, the attributes that are numerical. In that case, a pyDatalog function is used, which is a predicate with a parameter that can have a value (quite similar to Python's built-in `dict`). So, for example, the attribute number of samples with a value of 500 in pyDatalog will be represented as the fact `number['sample'] = 500`. This is useful because then the function's value can be used any number, most importantly to be compared to other numbers, something which a normal predicate parameter is unable to do. For the Data object of the problem, the idCode is used as a parameter and the value as the parameter. If the value should be a number, then the idCode should be 'number of ...'.

The *solution* is a bit more complex, since it has a much more complex structure. There is the Algorithm to be represented, the DataMiningStep and the string with other relevant information. The latter is actually the easiest to represent, as a predicate named aditionalInfo with one parameter, which is the additional information that should be presented to the user. The other recommended steps are represented similarly to the other recommended information: a predicate named otherSteps with three parameters, first the name, the second is the solution, and the third is the the details. Then there is the Algorithm which has a name and a list of hyperparameters. So there are 2 predicates to represent the Algorithm. First is recommendation with one parameter which represents the algorithm name. The second is hyperparameter

which has 4 parameters. The first parameter represents the algorithm it is meant for then the second is the name, the third is the value and the last one represents the details of the hyperparameter.

In order to transform the Problem into pyDatalog facts, it is a simple need to state the facts. It is also needed to store what facts have been added so that after the recommendation process is finished, those facts need to be retracted, so as to not interfere with the next requested recommendation. In order to transform the facts into a *Solution*, the knowledge bases need to be queried, process which will make use of the inference engine. a querry in pyDatalog is simply the call of a predicate with some parameters. If any of the parameters is a variable then it will return the values for the variable parameters for which the predicate is true. Otherwise it will return True or False, if the fact can be inferred from the knowledge bases or not. For example, in the querry `hyperparameter('SupportVectorMachines','kernel',X,Y)`, X and Y are parameters, so the querry will return all (X,Y) pairs for which the predicate is true, that is, all the recommended kernel for the support vector machine algorithm. We need four different queries:

- one to find all the recommended *algorithms*,
- one to find all the recommended *hyperparameters* for each recommended algorithm,
- one to find all the recommended *other data mining steps*
- one to find all the *other information* that might be relevant

The recommender performs all these queries, and then transforms the gathered data in a Solution object which is the returned to the `RuleController`.

Finally, the rules need to be written in a Prolog-like syntax. An algorithm is recommended if there is some data about the problem presented (the recommended predicate with that algorithm as parameter becomes true if some facts that represent data about the problem are true). And in a similar way all the other rules need to be defined.

The *information extractor* works mostly in the same way as the recommender, simply with other predicates, rules and queries. The idea is that the information extractor receives a Problem object containing some data but also returns a Problem object, that has a modified list of Data objects. The new Data objects in the list have a name an `idCode` and `possibleValues`, but don't have a value, so they act as questions for the user.

In the fact base, we have all the possible data that might be asked from the user. There is the 'question' predicate that has two parameters, first is the `idCode` of the Data and the second is the 'name' of the Data which is actually the question that will be presented to the user. Then there is the `possibleAnswer` predicate that has two parameters: the first is the `idCode`

of the Data and the second is one of the possible answers that Data can have. It is important to note that the `idCode` used here as a parameter is used in the recommender as a predicate, so when writing rules in the recommender or facts for the information extractor, it is important to be aware of the other and coordinate them. Possibly in a GUI aided management of the knowledge bases, this step would not be needed. For rules, the information extractors has one predicate, `askThis`, which takes one parameter, the `idCode` of the Data.

The information extractor's main role is to update the `Problem` with new `Data` objects which stand as questions. The process goes as follows (in pseudo-code):

Algorithm 1 Information Extractor

```
function FindNewQuestions(Problem givenProblem)
    facts ← formalizeToFacts(givenProblem)
    idCodes ← askThis(facts) // the ids of the questions to be asked
    for idCode in idCodes do
        name ← name(idCode)
        possibleAnswers ← possibleAnswers(idCode)
        Data newQuestion ← new Data(idCode, name, possibleAnswers)
        givenProblem.add(newQuestion)
    return givenProblem
```

Case-Based Reasoner The second intelligent method we implemented is the case-based reasoner. In our implementation the case-based reasoner works as follows: the case base is filtered depending on some of the data of the problem received, so only a subset of the case base remains (called eligible cases), then these cases receive a score, based on the similarity between their problem and the given problem, then, based on the score, a solution is created from the best cases found. Thus the task was to define a filter, a similarity function between two problems and a method to generate a Solution from the cases with the best score.

In order to filter the case base, we thought that the only applicable method is to filter depending on the goal. So only the cases which had a Problem with the same goal as the given problem can be eligible cases. Perhaps there might be better methods to filter the case base, based on combinations of attributes of the problem, but the definition of such filters requires more expert knowledge than we had at the time of the implementation of this first prototype.

The biggest challenge is to implement a good similarity function. What attributes should be given a priority over the other? Thus, given the knowledge we had access to in this first phase, we decided that the data type should have the biggest weight. We decided this because knowledge about the task the machine learning problem is trying to solve can, most of the times, improve the

efficiency of the machine learning. Then, the second most important features are the number of samples and the number of features, as different machine learning algorithms behave in different ways depending on the proportion of the samples to features, or some algorithms perform really well only with a small number of samples, while other algorithms perform really well with a big number of samples. Then the other features that we thought would weight a bit more than the rest are the fact if there are time or memory restrictions. That is, if the training time needs to be short (e.g. for the cases where there is need for online training) and if the algorithm has at its disposal whatever amount of memory, or if the machines the training is performing on are less equipped in terms of hardware. We took this into consideration because there are some algorithms that perform really well, but are big resources consumers, so, if there are not many resources available, other algorithms should be preferred. Then, all the other features are scaled the same. The score for every case goes from 0 to 1, but the maximum is 0.5 if the data type is different.

As the method to generate the solution, we chose simply to take the Solution of the case with the biggest score. It may not be the best method, but it is hard to automate the process of finding the best part of multiple solutions, and build another one out of them. One idea for a future work would be to have solution generation aided by a rule-based system, in where there are rules about what part of a solution is better in which case. This would be hard, because finding such rules is far from trivial. Another ideea would be to have a supervised algorithm to learn to generate a solution from the first best three cases. However, this requires a dataset of data containing a problem, three not-so-good solution and a better solution created from those three solutions. And such a dataset seems extremely hard to create. Thus, for the first prototype, we decided to go for the simple 'best case's solution'.

Solutions Combiner It is tightly linked to the other intelligent methods, so we will give some details about it in this subsection. In order to combine the solutions, we needed to combine the three different parts of the solutions independently: the recommended algorithms list, the recommended other data mining steps list and the other relevant information.

The latter is the easiest, as the two strings can be concatenated and all the relevant information will be available in the final solution. The other data mining steps list is, similarly, quite easy. The lists can be simply concatenated, we just have to make sure to remove the duplicates, which are the steps with the same name and value. We suppose that having the same name and value will have mostly the same information in the description, so we will just keep the one with the longer description, as we suppose it is the one that gives most details.

With the algorithms more questions arise. We have decided to display on the Graphical User Interface at most three recommended algorithms. We made this decision because we thought that, if we display too many recommended

algorithms to the user, then this defeats the purpose of the system itself, as it was specifically designed to help the user choose between the myriad of different algorithms and hyper-parameter combinations, so if it recommends many algorithms it is not really useful. Even if, in another implementation, it would display all the recommended algorithms, in order to avoid the previously described problem, these algorithms would have to be ordered in some way, or given a score or something similar. So the challenge is to find a method to rank these algorithms.

For the Case-Based Reasoning solution, we can suppose that the order in which they are entered by the user or the administrator is the order of best to worst, so these algorithm can be considered to be already sorted. On the other hand, the algorithms that are in the rule-based recommender solution are in a random order. With our current expert knowledge we could not find any kind of heuristic to just rank some algorithms and their hyperparameters by their efficiency. So, we decided to try an heuristic that seemed to be in favor of the more subjective purpose of our proposed system of giving support instead of going for efficiency of the recommended algorithm: we order them by the amount of details there are in the algorithm. This is because we think that more details about an algorithm the easier will be for the user to understand and to choose, rather than less information which might not help as much. Also, we so this step after we combine the same algorithms. This is quite a risky step, as sometimes there are two different combinations of hyperparameters that the user should try, but other times this helps to remove redundant information and complement information from one source with the other. We decided to go for the combination of the algorithms also because this system was based on the idea of combining the two different recommendation algorithms, so we believe that it may result in better or more exact recommended algorithms.

4.3 About Data

Until now we have talked about how the proposed system works in this specific implementation and how it moves around and processes data: asking new questions about the given problem, comparing the problem with already existent cases and inferring facts about the solution using facts about the problem and some rules. All this relies heavily on the data that the system uses: already existent cases in the case base and rules and facts for the rule-based recommender and the information extractor.

We have previously mentioned that this kind of data is to be entered by researchers or experts, as we believe that this kind of data needs expert knowledge in the domains of machine learning and data mining. On the other hand, this other data can be entered by a system administrator that extracts the data from different sources, such as journals, cheat sheets or question and answer websites (as we mentioned in chapter 2). And this 'administrator' role we took as we implemented the system, in order to enter some initial data in

the system.

The gathering of initial data can be quite time consuming, so, for an initial test of the system we decided to initialize the system data only with some basic data so to test if the functionalities are working properly and if the recommender works.

For the rule data, we thought that data from cheat sheets should be enough for a first prototype. We have gathered most of the data from the Scikit Learn cheat sheet. Although some straight-forward rules can be extracted from the cheat sheet, it is not trivial to represent them in the pyDatalog logic formalization. There is need to extract what are actual problem data that seems to make the difference in what algorithm or hyperparameter to use. Then we had to extract what are the actual recommended algorithms and hyperparameters, as the terms presented might be slightly different than the actual used algorithms. For example, in the Scikit Learn cheat sheet they mention the SVC and the SVR algorithms, while they are both Support Vector Machines, but one used for classification and the other for regression. So these cases have to be discovered and taken into account, and the difference between the SVC and SVR has to be made through the hyperparameters of the Support Vector Machine.

In addiction with the rules extraction from the cheat sheet, we also looked through the details of the recommended algorithms (Scikit Learn's website offers details of every algorithm mentioned in the cheat sheet) in order to extract more information, maybe even other recommended data mining steps (for examples for some algorithms it is recommended to scale the data first) or other relevant information (such as a list of non-linear kernels for the Support Vector Machines - yes, there might be a recommended kernel, but maybe that one is not he best one for the user's task and maybe having the information of other kernels will help to find the better solution).

For the case base, it is a bit more tricky as there is no repository containing data mining problems people have had at some point and their solutions. This information can be found in articles, machine learning competition results and other such places. One problem with extracting cases from articles is that the problems in the articles are very specific, so there would be need for many such specific cases to cover a more general kind of problem. Finding enough articles on a specific topic, that treat different aspects of that topic and hive different solutions is not trivial.

For the prototype first we decided to only search cases only for some types of problems. We decided to go with classification problems, as they are one of the more numerous type of problem and there is quite some research done in classification problems of many kinds. Still, for the first tests we decided to extract cases from the study of [5] where there is a review of multiple classification algorithms. We decided to do so because in this study there are

several types of classification algorithms used and the datasets used consist of real data and are of various types (images, biological, multivariate, etc.), and every classification algorithm was quite thoroughly tested with many different hyperparameter combinations. Thus, we could extract a case as follows: the problem had the details of the dataset given and the details of the experiments (e.g. using two classes), and the solution was created combining the details of the best two or three algorithm-hyperparameter combinations.

5 Experimental Evaluation

There is no simple objective way to evaluate this system. This system was design to help different kind of non-experts to choose a machine learning algorithm. It was not designed to give exact recommendations, or to give the best algorithm. The system is better if it can help target user to more efficiently solve their data-mining task. In order to properly evaluate the system, an experiment similar to an *usability test* would need to be performed.

Some potential users would have to agree to partake into an experiment regarding this new system. Then these potential users will be provided with the version of the system that we want to test. It is important that the users would do their normal data-mining tasks as usual. They will have to be monitored in some way while using the system, maybe by completing different surveys while and after they finished their data-mining task, and the comparison of their found solution and the system recommended solution. After a period, the data is gathered and analyzed. Only then we could really assess the evaluation of the system, as it is crucial to know how the system helped the users achieve their goal. We were not able to execute such an experiment because of lack of users ready to take the test.

A more intuitive approach would be to have some cases that are not in the case base and to enter the problem into the system then compare the output to the existing best output. This may look similar to usual testing, but the fact is that this is not a case of matching the output with the ground truth, as it usually is. The actual best solution for the problem may be very specific and even unique. The output of the system is meant to guide so that the best solution is found faster. Thus, the comparison of the real solution with the recommendation needs to be done manually and to assess how good would be the recommendation in order to find the real solution. An automated system could be put in place, maybe to score the similarity of the two, but it would be at best an approximation of the performance of the system. Also, when measuring the performance of the system, it is needed to take into account what is the current state of the knowledge bases and the case base. For example, if the case base contains only cases about prediction and classification of images, and the same the knowledge bases, then it would not make sense to try to asses the system's performance with cases with a different scope (such as dimensionality reduction of text) as it will obviously give bad recommendations.

In the case of our first prototype, the rule base contains just some general rules about general data mining tasks and the case base contains only binary classification cases. We decided to leave one of the cases out for manual testing. We entered the problem as the input, and received the recommendation. The first recommended algorithm was similar to the best solution (AdaBoost with Decision Trees) and the second recommended algorithm was similar to the third best algorithm (SVM with linear kernel). We decided that it was a fairly good result to continue with the project, and further testing will be done when there will be more cases and rules added to the case base and knowledge bases.

6 Sustainability Analysis

According to M. Adil Khan, a senior advisor of UNDP (United Nations Development Programme), in [17], sustainability may be defined as "the ability of a project to maintain its operations, services and benefits during its projected life time". Thus, a sustainability analysis is the identification of the factors that influence this ability of the project and asses to what degree they do it negatively or positively.

However, on the sustainability of software, in [31] it mentions that there is not yet an agreed definition on what software sustainability should refer to. Thus there are different definitions and practices that individual groups and organizations apply, sometimes even "holding diametrically opposed view". The problem is that the software sustainability can be viewed from many different point of view such as the software artifact perspective, or the software development sustainability or considering software sustainability a non-functional requirement [31]. For this reason we will rely on the more grounded, albeit also more general definition of project sustainability as presented in [17]. In [17] there is a list of items that describes what should an sustainability analysis contain: Relevancy, Acceptability, Economic and Financial Viability, Environmental Sustainability, Implementation and Monitoring Strategy and Post-implementation operation and maintenance.

Relevancy . This has more to do with the fact that the project may be done in the context of a bigger organization or a government, so the relevancy is tied to the priorities such organization or government may have, and if this project is relevant enough to be sustainable (so that it will not be discarded before the finishing of its projected life time for a project that is more relevant). As we are not working under a bigger organization or government this point of sustainability can be viewed as not influencing negatively the project.

Acceptability . It is tied to the degree of acceptability of the project by the community. This may be a relevant factor if the community might be affected by the development of this project or if would be a controversial project. It is quite clear that no one will be affected by the implementation of this project, as it is not funded by the government or any other organization, and in our point of view it is not a controversial topic so that the community might hesitate to accept it, because of this reason.

Economic/Financial viability . This obviously is linked to the monetization of the project. We have not taken in consideration this fact, as our main purpose for now was to prove that the concept of this IDSS is viable and could truly help non-experts with their data-mining tasks. However, if it would become a stand-alone web application that would have enough users,

we would think that there should be a good enough monetization method so that the project is sustainable.

Environmental Sustainability . This relates to the impact on the environment of the project. The environmental impact of a piece of software is related to the hardware it is stored and runs on [20] as the hardware web and large applications run on (data centers) are large consumers of energy which have an impact on the environment [7]. With our tests, the prototype is not a big consumer of resources. We did a test with 1 million mock cases on the case base, and even then, it only took several seconds to finish and it didn't use more than 40 MB of memory. While the finished project will be more complex and use more resources than the prototype, we don't think it will use enough resources to make any real difference on how much energy the data center it will be stored on uses. Thus, it will not have a big impact on the environment.

We wouldn't expect the cost of the project to be very high. The man hours needed for the software development should not be much more than needed for the implementation of the prototype (detailed in chapter 7) as the finished system only needs to be transformed into a web-based application. We expect that most of the work would be needed to further develop the GUI (as mentioned in chapter 8 - future work) and the adaptation of the case base to a data base (instead of a serialized file).

On the other hand, we expect that the task of initial data gathering to be much more demanding, requesting several times more man-hours than the development time. That is because to be usable by the user a large amount of data is needed. And while the system is designed to grow while it is used, still there is need for an initial data-base that can satisfy the needs of users.

As mentioned before, we do not expect the hardware requirements to be excessive, as the system is designed to be lightweight. While run from a bigger server, it should not use many resources. Still, this needs monitoring as hardware resources consumption might grow faster with a growing user-base, as more user request recommendations at the same time. Overall, we do not expect this to be costly.

We would expect that the most cost would be generated by the management of the case base, rule bases and fact base. Even though they should be managed by the community, there should also be some official administrators to complement the additions of the community as well as to provide management of the knowledge-bases while the community is not big enough to provide proper management. Thus, we expect a high cost to manage the knowledge bases especially at the beginning.

Overall, we do not expect the implementation cost to be high, instead we expect the maintenance cost to be higher, until a proper community is formed.

Implementation and Monitoring strategy . This is tied to the resources needed to implement the project. As we said before we have not taken into

consideration how to create and economically sustain the finished project, so there is not yet an implementation and monitoring strategy. This being said, we consider that the prototype presented here works acceptably and the only resource challenge that we would have to put the project as a real useful service would be the gathering of enough expert knowledge.

Post implementation operation and maintenance . We see that, in software, this step refers to how well could the project and prototype perform as a real-world service. That is, how would our project sustain the challenges of scalability and continuous management and support. As mentioned before, we did a test with many mock cases in the case base (1 million) and the prototype did not have a big problem with them. The recommendation took from 10 to 15 seconds, but we consider that this is not a big impact on this kind of service. Of course, the actual project will perform differently, but with the tests done on this prototype, it seems that the project will not have a big problem with scalability. The time complexity of the actual CBR recommendation algorithm is $O((7 + d)n)$ where n is the number of cases and d is the average number of other data per case. The spatial complexity is $O(2n)$.

On the other hand, the management of the system is not so easy to do. Not the management of the code would be a problem, but the management of the case base and the knowledge bases. We as administrators could not possibly do it, so there is need for experts and researchers to actually manage the system. This will happen if the system becomes used by many people and experts are incentivized to manage this system because it actually helps other people. This relies completely on how would such a system be received ad if it would be used.

7 Temporal Planification

It is important for any project to have your workload planned, in order to be able to complete it in time. Thus, we had our own plan in completing this master thesis project, which is summarized in a Gantt Chart that can be seen in Figure 10. This chart shows the tasks we defined, which are needed to complete the master thesis, and the time we dedicated to each task.

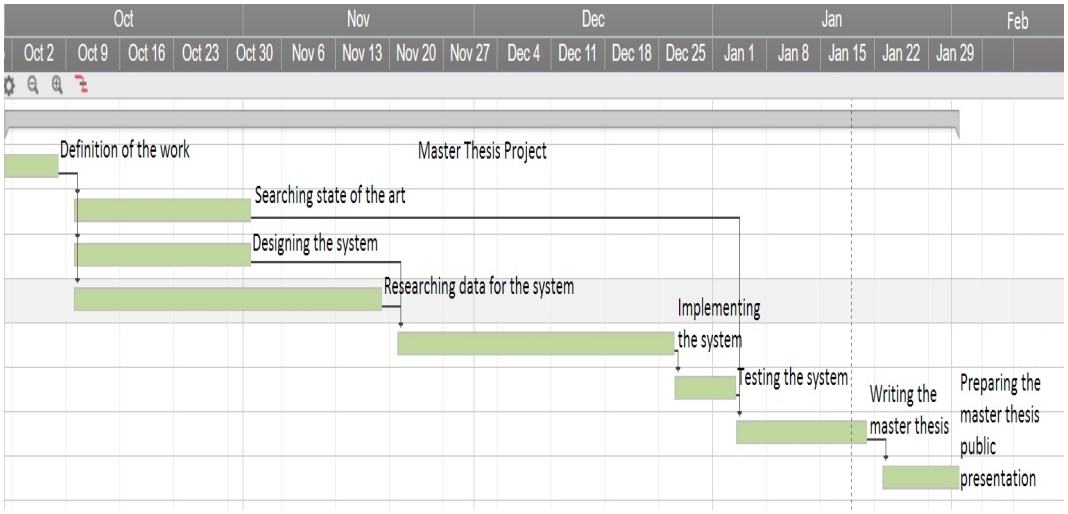

Figure 10: Gantt chart of the work plan.

The first thing that we needed to do was to explicitly define the work, as without a good definition, the project cannot be done. We planned and managed to finish this task in 6 days.

With the work ahead clearly defined we could now work towards implementing and testing our prototype, and towards writing the report. The report is dependent in having the results but parts of it were not, most notably the search for the state of the art (which, quite on the contrary, could help us with the implementation). We decided to finish the search for the state of the art as early as possible, with a limit at the 1st of November, because it was linked to the definition of the work and seemed more easy at that point.

Along with searching for the state of the art, we decided to also design the system and search for data for the system, so that the implementation can start as soon as possible. Initially we planned to do all three of them in the same time-frame, that is, before 1st of November. But the search for data for the system proved to be more challenging than foreseen, thus we had to lengthen the planned time for the data research with two weeks, to be sure we have the at least the data needed to be able to properly test the system.

Having the necessary data and the design of the system we started implementing the prototype. We left only five days for testing, as the system

is supposed to be quite lightweight, therefore testing it should not take too much time. We planned to finish the implementation and testing before 1st of January, but it took slightly longer than expected, so we only finished on the 4th of January.

Finally, we planned to finish the master thesis report until the 20th of January, which was the official deadline, and then to prepare for the oral presentation until the end of January, when the oral defense of the thesis takes place.

8 Conclusions and Future Work

In this work we have presented the concept of an Intelligent Decision Support System for Machine Learning Algorithm Recommendation and proposed a prototype of such a system. In our experience, there is a lack of tools for non-experts to help them choose between the many methods available to solve machine learning, and in particular data mining, tasks. Thus we had the idea to design a system that would help with these processes. We have presented this design, which is centered around the combined use of the Case-Based Reasoning and Rule-Based Reasoning, for the recommending process, while also trying to make the system easy to use and manage, for example, by using an information extractor; while insisting on the fact that the system is not designed to give the best possible algorithm or solution, but to help finding it. We have presented a prototype of such a system, and the implementation details of the two recommender algorithms. The preliminary testing of the prototype has shown it to be a promising tool.

Future Work. The most obvious future work would be the continuation of the development of the system from the existing prototype, to a real usable service. We have also mentioned throughout our work some of the many improvements that can be made to the system. First the Graphical User Interface would need to be greatly improved, as the one in the prototype is quite simpleS. It needs to become more user friendly, and there are two major functionalities to be added to it: a better (or maybe an actual) knowledge bases manager, that with which one can add and manage rules and facts through the GUI and not through a text editor; and improve the manager of the case base, because as there will be an increasing number of cases, a simple list is inefficient for managing, so functionalities such as filters, search and maybe tagging of cases need to be added, as well as faster case base formalisms.

Beyond the simple improvement of the prototype, there is plenty of space for improvement of the intelligent methods. The case-based reasoner might be changed to be an adaptive case-based reasoner, so that a part of the management of the case base is done automatically. Also, the case-based reasoner is using a very simple method for adaptation of the solution. It only uses the solution of the case with the best score. There might be a good future work in researching how can several best solutions be combined in order to get a better solution. Some ideas were using a learning algorithm (such as neural networks) using a rule system or maybe to use online learning (to adapt the combination algorithm every time a user introduces a new case by comparing the given case and the recommended one). The combination of the rule-based recommender solution and the case-based reasoner solution might also need improvement, for example, to make an algorithm to decide when to combine algorithms so that theirs respective details are added and the recommendation is better, and when to keep them separated, so that they represent different algorithm-hyperparameters combinations of the same algorithm.

References

[1] *What are the most important machine learning algorithms?*, Quora website, (2010). Website Link.

[2] *How can i go about applying machine learning algorithms to stock markets?*, Stack exchange website, (2011). Website Link.

[3] D. W. Aha, C. Marling, and I. Watson, *Case-based reasoning commentaries: introduction*, The Knowledge Engineering Review, 20 (2005), pp. 201–202. Full Text Link.

[4] D. Bridge, M. H. Goker, L. McGinty, and B. Smyth, *Case-based recommender systems*, The Knowledge Engineering Review, 20 (2006), pp. 315–320.

[5] R. Caruana and A. Niculescu-Mizil, *An empirical comparison of supervised learning algorithms*, ICML '06 Proceedings of the 23rd international conference on Machine learning, (2006), pp. 161–168. Pdf Link.

[6] W. Cheetham and I. Watson, *Fielded applications of case-based reasoning*, The Knowledge Engineering Review, 20 (2006), pp. 321–323. Pdf Link.

[7] P. Desmond, *Measuring the full environmental impact of your data center*, Schneider Electric Blog, (2013). Website Link.

[8] P. Domingos, *A few useful things to know about machine learning*, Communications of the ACM, 55 (2012), pp. 78–87. Pdf Link.

[9] M. Feurer, A. Klein, K. Eggensperger, J. T. Springenberg, M. Blum, and F. Hutter, *Efficient and robust automated machine learning*, NIPS'15 Proceedings of the 28th International Conference on Neural Information Processing Systems, (2015), pp. 2755–2763. Pdf Link.

[10] C. Giraud-Carrier, *The data mining advisor: meta-learning at the service of practitioners*, Proceedings of the Fourth International Conference on Machine Learning and Applications, (2006), pp. 113–119. Pdf Link.

[11] C. Giraud-Carrier, *Metalearning - a tutorial*, (2008). Pdf Link.

[12] J. M. Hernansaez, J. A. Bot, and A. F. Skarmeta, *Metala: a j2ee technology based framework for web mining*, Revista Colombiana de Computacion, 5 (2004). Pdf Link.

[13] A. Holt, I. Bichindaritz, R. Schmidt, and P. Perner, *Medical applications in case-based reasoning*, The Knowledge Engineering Review, 20 (2005), pp. 289–292. .

[14] E. B. U. Islam, *Comparison of conventional and modern load forecasting techniques based on artificial intelligence and expert systems*, IJCSI International Journal of Computer Science Issues, 8 (2011), pp. 504–513. Pdf Link.

[15] J. Joseph, O. Sharif, A. Kumar, S. Gadkari, and A. Mohan, *Using big data for machine learning analytics in manufacturing*, (2014). Pdf Link.

[16] A. Kaklauskas, *Biometric and intelligent decision making support*, Intelligent Systems Reference Library, 81 (2015). .

[17] M. A. Khan, *Planning for and monitoring of project sustainability: A guideline on concepts, issues and tools*, produced under the UNDP supported results-based Monitoring and Evaluation activity at the Monitoring and Progress Review Division of the Ministry of Plan Implementation, (2000). Website Link.

[18] A. L. Kidd, *Knowledge acquisition for expert systems: A practical handbook*, (1987). Google Books link.

[19] J. U. Kietz, F. Serban, A. Bernstein, and S. Fischer, *Data mining workflow templates for intelligent discovery assistance in rapidminer*, Proceedings of RCOMM'10, (2010), pp. 19 – 26. Pdf Link.

[20] K. Langborg-Hansen, *The environmental impact of running software*, Schneider Electric Blog, (2013). Website Link.

[21] G. Luo, *A review of automatic selection methods for machine learning algorithms and hyper-parameter values*, (2016). Pdf Link.

[22] M. Lutz, *Learning python*, (2008). Pdf Link.

[23] J. Qiu, Q. Wu, G. Ding, Y. Xu, and S. Feng, *Using big data for machine learning analytics in manufacturing*, EURASIP Journal on Advances in Signal Processing, (2014). Pdf Link.

[24] L. Rendell, R. Seshu, and D. Tcheng, *Layered concept-learning and dynamically-variable bias management*, Proceedings of the 10th international joint conference on Artificial intelligence, 1 (1987), pp. 308–314. Pdf Link.

[25] M. M. Richter and A. Aamodt, *Case-based reasoning foundations*, The Knowledge Engineering Review, 20 (2005), pp. 203–207. Pdf Link.

[26] T. Roth–Berghofer and I. Iglezakis, *Six steps in case–based reasoning: Towards a maintenance methodology for case–based reasoning systems*, Professionelles Wissens management: Erfahrungen und Visionen (Proceedings of the 9th German Workshop on Case-Based Reasoning (GWCBR)), (2001), pp. 198–208. Pdf Link.

[27] S. Sahin, M. Tolun, and R. Hassanpour, *Hybrid expert systems: A survey of current approaches and applications*, Expert Systems with Applications, 39 (2012), pp. 4609–4617. Pdf Link.

[28] F. Serban, J. Vanschoren, J.-U. Kietz, and A. Bernstein, *A survey of intelligent assistants for data analysis*, ACM Computing Surveys (CSUR), 45 (2013), p. 31. Pdf Link.

[29] R. Striz, *Metalearning for data mining and kdd*, (2012). Pdf Link.

[30] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, *Auto-weka: Combined selection and hyperparameter optimization of classification algorithms*, KDD '13 Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining, (2013), pp. 847–855. Pdf Link.

[31] C. C. Venters, C. Jay, L. M. S. Lau, M. K. Griffiths, V. Holmes, R. R. Ward, J. Austin, C. E. Dibsdale, and J. Xu, *Software sustainability: The modern tower of babel*, CEUR Workshop Proceedings. RE4SuSy: Third International Workshop on Requirements Engineering for Sustainable Systems, (2014), pp. 7 – 12. Pdf Link.

[32] D. H. Wolpert and W. G. Macready, *No free lunch theorems for optimization*, IEEE transactions on evolutionary computation, 1 (1997), pp. 67–82. Pdf Link.

[33] T. Yu, T. Jan, S. Simoff, and J. Debenham, *Incorporating prior domain knowledge into inductive machine learning*, (2007). Pdf Link.

[34] R. Zucker, J.-U. Kietz, and A. Vaduva, *Mining mart: Metadata-driven preprocessing*, (2001). Pdf Link.