Master research Internship

Bibliographic report

Automatic planning of Data Science workflows in Answer Set Programming

Domain: KDD process - Planning

Author:
François Laferrière

Supervisor:
Alexandre Termier
Torsten Schaub
LACODAM

Abstract:

Contents

1 Introduction

The progress made in hardware technology has made it possible to digitize more and more phenomena. The storage of the these digital information led to the growth of huge databases. The large amount of information contained in these databases has the potential to allow to make new discovery, or to optimize industrial systems. However, the size of these databases prevent them to be analyzed manually. There is thus a need of automatic methods. The Knowledge Discovery in Databases (KDD) aim to solve this problem, however it is a difficult task. It requires to use many different data processing and mining operators. The KDD process involves to build a workflow combining several data transformation and mining operators. Currently, the composition of these worflows must be made by a human, who consequently needs solid knowledge about the KDD process and the operators it use. There exists many data mining operator that can possibly be mixed into millions of different workflows. It is thus easy to understand that even the most experienced data miners can be overwhelmed by the complexity of the task. Being able to assist users during the KDD process is a major challenge nowadays. Our goal is to design a system able to provide assistance during the KDD process in order to relieve the stain of the user, and if possible to permit novice in data mining to achieve the KDD task. A system able to guide the workflow construction process would provide an invaluable help to data miners. Such systems are called Intelligent Discovery Assistants (IDA). Some already exists, but they generally only provide partial guidance to scientist. None of them fully covers the data mining process.

We want to design an IDA able to automatically design workflows that can solve difficult tasks such as pattern mining. The objective is to build a system as automatic as possible to be able to discharge the user of as much work as possible. The best approach to design such a system seems to be a mix of planning and meta mining approach. For planning part we will use *Answer Set Programming* which is a declarative constraint programming language based on non-monotonic logic well suited for domain representation.

This rapport is structured as follows. We first review in section 2 the existing IDAs. In section 3 we detail one of the most advanced systems and analyzes its limits. After identifying the planning as one of these limit, we present in section 4 different planning methods that could be useful to design an IDA.

2 Intelligent Assistants for Data Analysis

This section is dedicated to an overview of IDA's state of the art. In section 2.1 we describe the KDD process. Section 2.2 presents the different types of IDAs and section 2.3 provide a comparison of these different types of system.

2.1 KDD process

The Knowledge Discovery in Databases process can be decomposed in five parts, selection, preprocessing, transformation, data mining and evaluation (Figure 1). In the selection step, the relevant raw data are selected and collected. They are then cleaned from noise, errors or missing values in the preprocessing step. Finally, the data are transformed to be adapted to the input format of data mining operators. Those steps can be for example composed of data sampling, feature selection, or extracting SIFT descriptor for images. The data mining step consist of applying a data mining technique to find models or patterns. Those techniques range from classifier such as SVM or neural

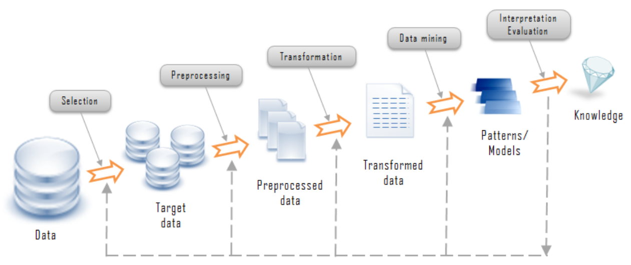

Figure 1: The KDD process

network to pattern mining techniques. The last step is to interpret or to evaluate the result of the applied technique. For the SVM example, the last step would be to calculate the precision and the recall.

Building a data mining workflow consists of finding which operators will be the most effective to solve each KDD step described above, with respect to the data and the scientist goal.

Today's knowledge discovery support propose such as Rapid Miner[1] more than 500 operators. Combining those operators can lead to several millions of possible workflows and make the knowledge discovery process very complex for inexperienced users. There is thus a need intelligent assistants able to support users in the workflow construction task.

2.2 Types of systems

IDAs that provide guidance for designing workflow can be divided into various categories that use different type of technique to advise the user. Serban et al. [6] define five different categories : the *Expert Systems*, the *Meta-learning systems*, the *Case-based reasoning systems*, the *Planning systems* and the *Workflow composition environments*

2.2.1 Expert systems

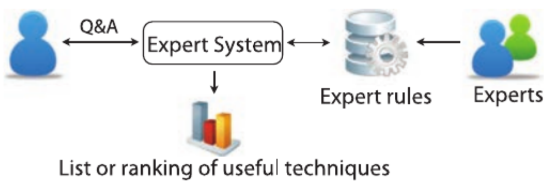

Figure 2: The general architecture of expert systems (credit to [6])

The first type of IDAs are *Expert systems*. Figure 2 provide an overview their architecture. Those systems regroup expert rules into a knowledge base.

Those rules are manually designed by experts. They define which techniques to use in with conditions. The system first ask questions about the problem to the user to find which rules to

[1]https://rapidminer.com/

apply. Then it generally returns a ranked list of techniques that it determined suitable to the problem.

Expert systems are build with a fixed size of techniques, they can only advise techniques for which rules were defined by the experts. However, the number of data mining techniques are in constant augmentation. Maintain an expert system up to date is a tremendous task. When new techniques are added to a system, an expert need to define new rules. It is easy to understand that this type of approach will quickly be overwhelmed by the number of techniques. Another problem of expert systems is that they only advise users on the choice of special data mining algorithm, but do not provide any help on the complete workflow design. Expert systems provide advise with respect to the given problem, but they do not consider the data itself. Taking into account which kind of data we are dealing with is important to find which techniques are the more suitable. This problem leads use to the next type of IDAs, the Meta-learning systems.

2.2.2 Meta-learning systems

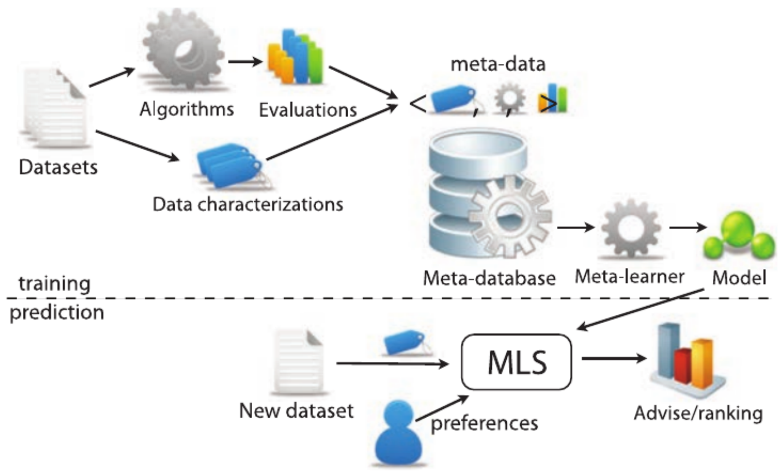

Figure 3: The general architecture of meta-learning systems (credit to [6])

Meta-learning systems (Figure 3) are based on the hypothesis that we can learn the relationship between meta-data about the datasets and the performance of different algorithms applied on those datasets. The system then uses this knowledge to predict which algorithm is the most suitable for a given dataset. A meta-learning system has two step. The first part is the training phase during which the system learn a model, and the second is the predictive phase in which the system decides which algorithm should be used on the data. The first step of the learning phase consists in extracting meta-data from a dataset. Those meta-data are a set of various measurable characteristics that characterize the dataset. We will not discuss here which type of characteristics should be used. After that, the algorithms are applied to the dataset, and the performance result (speed, accuracy, error rate...) are stored, linked to the characteristics of the dataset. This process is applied on all the training datasets. A model that defines algorithm performance on different types of data is then learned using the previously gathered information. This model is used in the predictive phase to rank algorithms with respect to their performance on a new dataset, based on the characteristics of this new dataset. Those systems often allow the user to define preferences on which type of performance he wants to favor (speed rather than accuracy for example.

We saw that the drawback of Expert systems is that new techniques are difficult to integrate into the system. This issue do not stand for meta-learning systems. When new operators have to be added to the system, the model is automatically updated by running those new algorithms on the training datasets. However, like expert systems, meta-learning systems only provide advise on a single step and not on how to build the entire workflow. The utility of these systems is therefore limited to cases where the preprocessing is already done.

2.2.3 Case-based reasoning systems

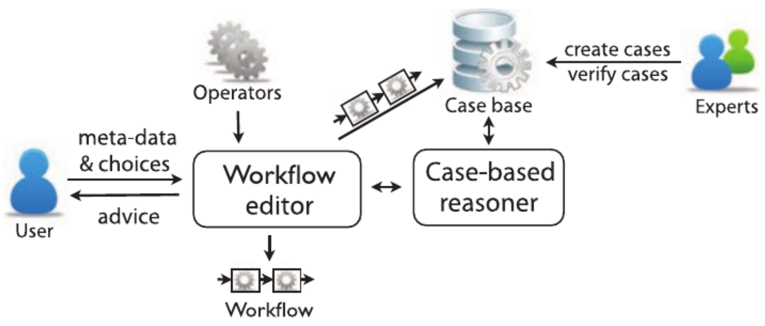

Figure 4: The general architecture of case-based reasoning systems (credit to [6])

Case-based reasoning systems (Figure 4) are quite similar to meta-learning systems in the fact that they both based on the use of previously executed techniques. Unlike meta-learning system that learns a model during a separate training phase, case-based reasoning systems store successful workflows into a *case base*. When a new problem is given by an user, the system will find to which problem in the case base the new one is similar using characteristic on the dataset, and the user preference. The system then return a list of workflows from the case base that performed well on those similar problems. After that, the user just need to select one of the workflow in the list and load it into a workflow editor to adapt it to the new dataset. Once a workflow executed, it can be added to the case base.

This approach has the advantage to provide a complete workflow to solve a new problem. However, even if the workflow can be appropriate to the given problem, it is rarely the case that it will entirely suit to the data. It is generally needed to modify the workflow, deleting or adding some operators, to allow the data to have the right format. The fact that the task of performing this step is left to the user is a first problem. But a second equally important is that this type of system does not always make it possible to verify the correction of a workflow (in the sense that it can be executed without any failure). This problem is addressed by the next IDAs type, the planning systems.

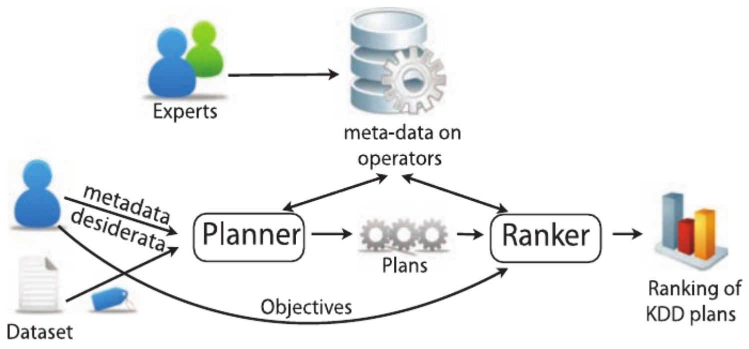

Figure 5: The general architecture of planning-based data analysis systems (credit to)

2.2.4 Planning-based data analysis systems

Previous systems use experience to advice users, but when a new input problem is different from everything that have already be seen, they cannot provide other support than general advice, and even less a valid workflow. To address this problem we need a system able to build from scratch entire workflows that are valid. This can be done by looking at the workflow construction problem as planning problem where the goal would be to extract something interesting from the data (build a model, extract significant patterns...). A plan is a sequence of operator that transform the initial data, run a data mining algorithm and evaluate the results. More explanations about planning are provided in section 3.

The first requirement of planning-based data analysis systems (Figure 5) is information about the inputs, the outputs, the preconditions and the effects (IOPE) of every operator. Secondly, system use the dataset characteristics and user preferences to build a planning domain description. With those information, all valid workflows can be build. However, this usually means hundreds of different plans that need to be ranked according to the user's objectives. To address this problem, the system need other information that IOPE, such as operators relative speed or accuracy. Most of the time IOPE are stored in hardcoded form in *ontologies*. Information about all the operators of a plan can be then combined to evaluate the resulting workflow.

The capacity that planning-based data analysis systems to ensure workflows correctness is very interesting. However, those system do not use knowledge from past workflows and have to start from scratch at each new problem. Furthermore, workflows performance are based on operators metadata (speed, accuracy...) taken independently from the dataset. Yet, those information is closely linked to the dataset characteristics. This often results in bad workflow performance estimation.

2.2.5 Workflow composition environments

The last type of IDA is different form the previous ones since workflow composition environments do not give advice on operators choice. Instead, they provide support to the user during manual workflow composition through a high-level scripting language that allows quick workflow design and execution, or through a graphical environment, where the dataflows can be dawn on a canvas. The general architecture of workflow composition environment is shown in Figure 6 Workflow com-

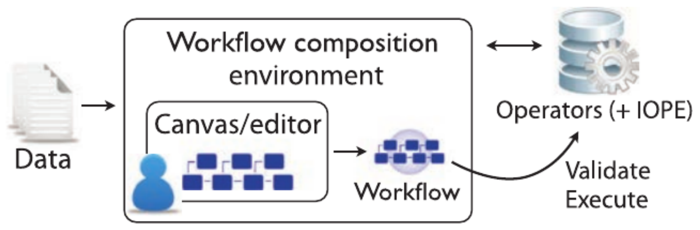

Figure 6: The general architecture of workflow composition environments (credit to)

position environments first provide a collection of algorithms, but can also propose some guidance to the user such as checking workflow for correctness before execution, metadata propagation or auto-wiring. They are also the only IDAs actively used.

2.3 Comparison

First, IDAs should be able to use all existing operators. Expert systems as well as Meta-learning systems only use data mining operators, and can't provide advice on other step of the KDD process. They are thus only useful if the user has already done the preprocessing. Other systems provide guidance on the all process, however, most IDAs only support a few techniques in each KDD phase and can't always be easily extended (new expert rules or new cases are needed). Workflow composition environment generally control hundreds of operators and thus cover a vast range of task. Planning systems can incorporate new operators only by adding the operators and their IOPE into their ontologies.

The second important point is the quality of the guidance. To provide pertinent advice, an ideal IDA should be able to learn from previous executed workflows and to use this experience for the operators choice of a new task. Expert systems use harcoded rules written by expert, but we saw that, as new techniques are introduced at constant rate, and new rules need to be define to integrate a new operator, it is unfeasible to maintain those system up to date. Meta-learning systems build predictive model on meta-data, and case-based system store successful workflows, assuming that they will perform well on similar data. They thus both make use of past workflows to improve the advice given to the user. They also both rely on metadata of datasets to measure their characteristics, and estimate their similarity. In the other hand, planning-based systems generate valid workflows, but do not use experience gained from previous task and start from scratch every time.

Each system type has to strengths and weaknesses. To build a ideal IDA, we should try to combine best aspects of each approach, which seems possible as they are generally compatible. An ideal IDA should be able to automatically generate all valid workflows and easily incorporate new operators as do planning based systems. It should learn from previous task and improve as new workflows are built, as do meta-learning and case-based systems. Finally, an ideal IDA should be able do execute itself workflows as generally allow workflow composition environments.

3 Using Meta-mining to Support Data Mining Workflow Planning and Optimization

A possible approach that would correspond to an ideal IDA as described in the previous section could be to generate workflows with a planner that would be guided in the operators choices by a meta-mining system. Nguyen et al. propose such a system in 2014 [5]. In that system, a meta-mining module analyses past experiments in order to extract a model that associate dataset characteristics with workflow descriptors. The model is then used to guide a planner during the workflow planning. This system is the only one to our knowledge proposing to automatically plan data mining workflows that are expected to optimize a given performance measure. What we attempt to do is very similar to this system. It allows us to understand where is the state of the art. For this reason, this section is dedicated to the detailed description of the system presented by Nguyen et al.

3.1 System Architecture

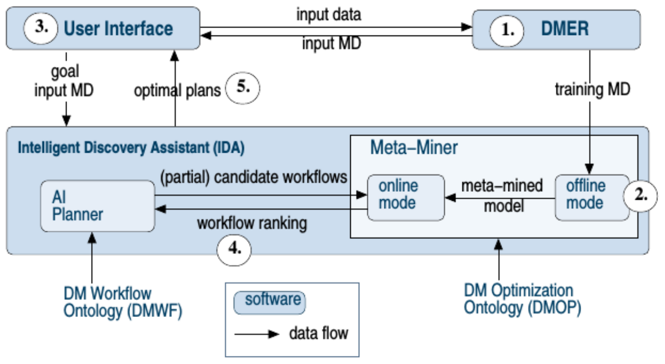

Figure 7: Architecture of the meta-mining system (credit to [5])

The system is split into 3 part: the *Data Mining Experiment Repository* (DMER), the user interface and the IDA as shown in figure 7. The DMER stores all resources necessary to learn models that will guide the planning. This includes training datasets, workflows and their performance results. The interface serves as a means of communication between the user and the system, to specify task instructions and input dataset. The IDA is the core of the system. It contains the meta-learner and the planner which interact to build optimal workflows.

The system operates in two modes, an offline mode during which the meta-mining model is learned, and an online mode during which the optimal workflows for a given problem are generated.

The first step in the offline part is to collect data-mining experiments to build the training base. These experiments are stored into the DMER. Once enough training data are collected, the meta miner can use these experiments to learn a meta mining model. This is done by extracting from the past experiments stored in the DMER characteristics that describe datasets, workflows and the performance of the workflows on the datasets. With these meta-data, the meta miner learns relations between the characteristics of datasets and workflows that have high performances by learning a heterogeneous similarity measure. This similarity measure reflects how well a workflow is expected to work on a dataset, i.e. if a workflow is expected to achieve high performance on a dataset, the heterogeneous similarity measure between this workflow and this dataset will be high.

Once the model is learned, the online part can start. The system directly interacts with the user. The latter can specify the input dataset and which data mining task to apply on it (classification, clustering...). Additional preferences, such as the number of workflows that should be planned, can also be communicated. With these information the planner can start the planning process. During this process, at each step, the planner generate all valid partial workflows. Those workflows are then passed to the meta-miner that use the learned model to rank them according to their expected performance on the dataset given by the user. The k best workflows are selected and the planning continues to develop them until the task is solved. In other words, the planner will first generate all valid plan containing only one operator (length 1). The meta-miner selects only the best candidates. The planner generates all valid plans containing 2 operators by adding a new one at the end of the best workflows selected by the meta-miner. The meta-miner selects only the best candidates among the workflows of length 2 and the planner will repeat the process with those workflows by adding operators until the task defined by the user is solved. Once the planning process complete, the best workflows are presented to the user, ranked according to their expected performance on the input dataset. This approach of planning is greedy as the selection of the best workflows is done locally. Some of the abandoned workflow might have led to globally optimal workflow. Letting the planner generate all valid workflow and rank them according to their global performance would always lead to the selection of the optimal workflows. However, this is not feasible as the number of possible workflows grows exponentially.

3.2 DM Workflow Representation

Nguyen et al. represent data mining workflow as hierarchical directed acyclic typed graphs (DAGs). The operators are represented by the nodes of the graph. The input and output of operators are represented by the edge between nodes. The hierarchical aspect is due to the fact that some nodes (operators) can contain sub-workflows. For example, the cross-validation operator will contain a training and a testing sub-workflow. Such operators are called *dominating* operators.

Formally, let O be the set off all available operators (Naive Bayes, SVMs, etc), including *dominating* operators. O represent all the possible nodes of a DAG. An operator $o \in O$ is defined by a name and the data types $e \in E$ of its inputs and outputs, as well as its sub-workflows if o is a *dominating* operator, E is the set off all possible data type that can take the input or output of all operators in O. $e \in E$ is a pair of nodes (o_i, o_j), that correspond to the inputs and outputs data types that are passed between operators o_i and o_j. It is the set of all possible edge within a graph.

Furthermore, operators that share the same input and output data types are grouped under a same abstract operator $\hat{O} = \{o_1, o_2, ..., o_n\}$. This abstract operator represent all the possibilities of choice among operators sharing the same syntactic characteristics. For example, the abstract $PredictiveSupervisedLearner$ operator will regroup classification algorithm such as SVM or $NaiveBayes$.

A DAG is represented by a pair (O', E') where O' is a subset of O, containing all vertices corresponding to the operators used in the workflows, and E' is a subset of E corresponding to edges of the graph. Edges of DAGs are directed. Therefore, an edge (o_i, o_j) implies that the operator o_i appears before the operator o_j. DAGs are acyclic, it is then possible to define a $topologicalsort$ of a DAG, which is a complete ordering of the nodes of a DAG. A topological sort of a data-mining workflow can be modeled as a tree by doing a deep-first search over its graph structure.

3.3 Workflow planning

The workflow planner uses the hierarchical task network (HTN) approach which can deal with primitive tasks, i.e. directly executable tasks, but also compound tasks. A compound task is a complex task composed of a sequence of sub-tasks that can themselves be primitive tasks, or compound task.

The planner extracts from the selected goal g in the set of goal G, the set T of all task that can address g. Then for each task $t \in T$, there is a set M of methods sharing the same data inputs and outputs than t, that can achieve it. Each method $m \in M$ define a sequence of operators or sub-task, which achieve m when executed in that order. The planner sequentially construct an HTN plan by recursively expanding task, methods and operators. Terminal nodes of the HTN plan correspond to operators, and non-terminals to the decomposition of task or method, or to dominating operators. All information about the set of goals, methods, task and operators including the inputs and outputs specifications are stored into an ontologie.

The overall grammar they define contains description of 16 tasks and around 100 operators. It is specified that these numbers are not limited by the planner capacities but by the modeling efforts that need the description of new tasks or operators.

During the construction of the HTN plan, the planner build many partial workflows. When those partial workflows are derived by substituting abstract operators with every operator they regroup, several thousands of workflows are generated. This is a problem as planning all those workflow can take very long time. It is even possible that the planning process never terminate.

To avoid exploring the all planning space, the planner is locally guided. The drawback of this greedy approach is that the global optimum might not be found. However, the potential reduction of quality of the planned workflow is offset by the considerable time gain that this approach allows.

A heuristic hill climbing approach is followed to guide the planner. Whenever an abstract operator $\hat{O}$ have to be substituted, the system need to know which of the operators $o \in \hat{O}$ are expected to achieve the best performance on the given dataset. More formally, let

$$C_l := \{w_{lo} = [w_{l-1} \in S_{l-1} | o \in \hat{O}]\}^{k \times n}$$

be the set of $k \times n$ partial candidate workflows of length l obtained by expanding an abstract operator $\hat{O}$. This set is generated by adding to workflows in S_{l-1}, that where selected in the previous planning step, one of the operators $o \in \hat{O}$. The new set of optimal partial workflows S_l is obtained by selecting the k best plan according to:

$$S_l := \{\arg \max_{\{w_{lo} \in C_l\}} \hat{r}(\mathbf{x}_u, \mathbf{w}_{c_{lo}} | g)\}^k$$

where $\mathbf{w}_{c_{lo}}$ is a binary vector that provide a propositional description of the workflow w_{lo}, $\mathbf{x_u}$ a vector describing the input dataset, and $\hat{r}(\mathbf{x}_u, \mathbf{w}_{c_{lo}} | g)$ is the estimated performance of the workflow described by $\mathbf{w}_{c_{lo}}$ on the given dataset described by $\mathbf{x_u}$.

3.4 The Meta-Miner

In this part we will describe how the meta-miner learns the model that will guide the planner during the planning.

3.4.1 meta-data and performance measures

In the system, the datasets are described with the help 150 numeric characteristics of 3 different types: the *statistical and information-theoretic measures* such as the number of instance or the class entropy, the *geometrical and topological measures* such as the maximum Fisher's discriminant ratio or the ratio of average intra/inter class nearest neighbour distance, and the *landmarking and model-based measures.* We will not further develop this point here, even if it may be important for building a pertinent model, as our interest lies first on the planning part.

The description of a workflow is done by a propositional representation of patterns present in the workflow. A data mining approach is used to extract frequent patterns over the tree representation of the workflows in the training base. The propositional representation specify the presence or not in the workflow, of the patterns extracted. Additional knowledges about the behaviors of algorithms that a workflow contain are also added in the workflow description. To give an example of behavior, a SVM is tolerant to irrelevant attributes, but not to missing values. The way in which the descriptions are made allows them to provide information about the behavior and relationships of the different operators in the workflow, which will be useful for the performance estimation.

For the same reason that for the dataset description, we will not further develop the workflow description approach here.

The performance estimation $\hat{r}(\mathbf{x}_u, \mathbf{w}_{c_{lo}}|g)$ is done with respect to the description $\mathbf{w}_{c_{lo}}$ of the workflow, and not only with the description of the operator candidate (the added operator). As said above, the description provides information about the relation between the different operators of the workflows. Thus, the ranking of the partial workflow is done with respect to the relations of the biases of the different algorithms within these workflows, and not only the sum of independent performance of isolated operators. To rank workflows with respect to their performance on a given dataset, each pairs of workflows are compared. If one of the two compared workflows has significantly better performance on the dataset, it gains 1 point and the other 0 point. Else, they both get half a point. The significance test is done using McNemar's test, with a significance level of 0.05. The final ranking is done by summing all the points a workflow gained, the higher the better.

3.4.2 Learning meta-mining models

The meta meta-mining model is constructed using two different strategies.

The first strategy learns homogeneous similarity measures. Similarity between datasets and workflows are learned independently. A first similarity measure considers two datasets to be similar if the relative performance of applying workflows to both of them is similar. In the same way, two workflows are deemed similar if, applied on a set of datasets, their relative performance are similar. During the planning, the similarities between the input dataset and all the training datasets are computed. Then, at each planning step, the meta-miner determine the similarity of each candidate workflow to each of the training workflows. The final ranking of a candidate workflow are estimated through a weighted average of the rank of all the workflows in the training base on all the datasets in that training base. The weights are given by the similarity between the input dataset and the training datasets, and by the similarities of the candidate workflow to each workflows of the training base. The first strategy use similarity measures learned independently of each other, and thus, does not take into account interactions between workflows and datasets characteristics.

The second strategy considers that certain type of workflows give better results on datasets with particular types of characteristics. A heterogeneous similarity measures that directly estimates the similarity between a workflow and the input dataset is learned. Briefly, the way heterogeneous similarity measure is learned can be summarized as performing a projection into the same space of the datasets and workflows description, and then computing a standard similarity/appropriateness measure in that space. During the planning phase, the heterogeneous similarity measure is directly used to rank candidate workflows.

The main advantage of this meta-mining approach is that the models used by the planner to rank workflows rely on the descriptions $\mathbf{w}_{c_{l_o}}$ of the workflows. Those descriptions are generalized over operators, and thus do not mention operators specifically. This allows the systems to plan workflows containing operators not present in the training base, as long as the IOPE and behaviors of these unseen operators are known. This characteristic permit to newly designed algorithms to be almost instantly integrated to the system, without needing to wait the training base to be enriched with new workflows. The system can thus easily be maintained up to date.

3.5 Experimentation

The authors present evaluation of the system on the data mining task of classification. The choice to only use this task is mainly made for practical reasons. Indeed, the classification has been extensively studied, and thus their is plenty of benchmark available. Classification task also allows to compare results of the systems with a ground truth, as this is a supervised task. Furthermore, the performance evaluation can easily be done using different measure such as precision or recall.

They compose a training base with 35 different workflows. Among these 35 workflows, 7 contain only a classification algorithm, and 28 are a combination of a feature selection and a classification algorithm. Those workflows are applied on 65 datasets, resulting in 2275 base-level experiments.

The system is tested in two different case, using a leave one apart strategy. In the first one, workflows are built using only operators seen in the training base, and in the second new operators are introduced for the workflow construction.

Briefly, the result they obtain using the heterogeneous measure seems good in the first case, but not when unseen operators are introduced. Performance of the system with homogeneous measure is never significantly better than the baseline they define.

We don't find pertinent to develop more the explanation of their experimental protocol, as they plan workflows containing at most 2 operators. Indeed the longest workflows they build only contain a feature selection and a classification algorithm. Some others contain only one operator, and thus can't be considered as workflows. We could still want to draw conclusions from the results. The experiment show that planning with unseen operators lead to bad workflows. However, this behavior could be expected because the system can make good use to new operators only if their characteristics are already well represented in the training base. The training base only contain a few operators, it is thus idealistic to think that characteristics of new one are well represented by those few operators.

Furthermore, the system is only tested on classification tasks. Other task such as pattern mining are much more difficult to evaluate, thus result obtained on classifications task cannot be extrapolated for those task.

To evaluate in a pertinent and sufficient way this system, a training base containing long enough workflows and various operators is needed. Those operators have to be well described in an ontology, and a way to evaluate workflows must be found for every data-mining task. The conclusion we can draw here is that evaluate such a system is a complex and time-consuming task.

3.6 Toward workflows for pattern mining

A problem to solve to be able to give a good evaluation of the system is first to find a way to evaluate the workflows. For a classification task, this problem is quite simple to solve because the task is supervised. We just have to use measure such as precision or recall. However, it is much more difficult to evaluate the result of a pattern mining task, where a ground truth does not exist.

Hanhijarvi et al. [2] propose a multiple hypothesis testing method assessing simultaneously the significance of all frequent pattern in a dataset. They use a randomization algorithm to sample datasets from the *null distribution*. Then, the intuition is that a pattern x is significant if the test statistic for this pattern is an extreme value in the null distribution. With this approach, they calculate for each pattern a pvalue that reflects the statistical significance that this pattern is frequent. We can easily imagine a way to use these pvalues to evaluate the results of a pattern mining algorithm. For example, a measure of performance could be the means of the pvalues of the k patterns with the bests pvalues. It would then be possible to evaluate workflows with respect to a pattern mining task.

The planner used in Nguyen et al. is quite simple as it only use a hierarchical task network. This might be sufficient for what they attempt to do, as they only solve classification task, and build very short workflow. However, we are more interested in the pattern mining task which is more difficult. The workflows our planner will aim to build will be more complex. Therefore, we need more complex planning techniques. In the next section we describe some planning methods that could help the planner to design complex workflows.

4 Planning

One of our idea is to build a system able to plan workflows without precise task specification. Often, the user only know that some relevant information are hidden in a dataset, but has no clue about what is interesting in these data. In this case, he cannot define a concrete task for the system to solve. Thus we want solver to be able to start planning without a well defined goal. Our idea is to let the system interact with the user during the planning, the goal being precised as the planning continue. For example, the system could generate result with different approach on small sample of the input dataset using sampling methods, ask the user which result he likes bests, and thus learn the type of result expected.

This kind of approach cannot be explored using only HTN planning. Their is therefore a need for different planning methods able, do deal with complex problem such as planning with incomplete knowledge or user preferences.

4.1 Planning techniques

The simplest formulation of the planning problem consists of finding a sequence of actions leading a world from an known initial state, to a final state, that satisfy goal conditions of a given goal. States are represented through fluent, i.e. properties of the world. To perform this task, a planner need to know the initial state which describe the world as it is before that any planned action is applied, the goal conditions, as well as the actions preconditions and effects. For example, a precondition to an action *cleanthekitchen* is to be *inthekitchen*, and an effect of this action is that the *kitchenisclean*. In the case of the system presented in the previous section, actions correspond to data-mining operators, and the initial state correspond to a description of the input dataset.

4.1.1 Action language

To build a valid sequence of action leading from the initial state to a goal state we need to be able to reason about a single action. This generally involve a transition function which, given a state of the world and an action, define in which states applying this action might lead the world. However, the space need to the explicit representation of this function is exponential in the number of fluents. The action description language B [7] allows to describe the precondition and effects of action on propositional fluent, and then define implicitly to transition function. This language use a set of proposition of the form:

$$causes(a, f, \{p_1, ..., p_n\}) \tag{1}$$

$$caused(\{p_1, ..., p_n\}, f) \tag{2}$$

$$executable(a, \{p_1, ..., p_n\}) \tag{3}$$

where f and p_i are fluent literal, i.e. a fluent t or its negation $/t$, and a is an action. The $dynamiccausallaw$ (1) represent the conditional effect of action a. It states that the fluent f always holds after executing a in a state where $p_1, ..., p_n$ hold. The $staticcausallaw$ (2) express that f always holds if $p_1, ..., p_n$ hold. Proposition of the form (3) states that the action a is executable if $p_1, ..., p_n$ hold. Once information about actions have been well described by the action language, the transition function is implicitly define in term of that description, avoiding to explicitly represent all the transitions rules.

4.1.2 Planning with incomplete knowledge

Planning techniques described until know only allow to plan under the assumption of complete knowledge about the world. However, to build more complex workflows, we need to face the case where information are missing. ROBOT introduce sensing action, which provide information about the world, but leave it unchanged. Such an action could for example asking information about the data to the user. The result of a sensing action is only known once the action has been executed. Hofmann propose to overcome this problem using Continual planning. Instead of creating a complete plan before execution, the continual planning approach intend to interleave planning and plan execution, thus the plan is constantly updated according to newly acquired knowledge. They define *assertions* which are actions representing conditional sub-plans which are dependent of unknown fluents. Assertions can't be executed, but they guarantees that its effects are achievable if the precondition are met. Imagine that at some point in a plan, the dataset is needed to have no missing values. One could want to delete the raw containing a missing value, if the the dataset is big enough but to replace the missing value with an estimation such as the means of other raws value, if the datasets is not large and losing raw is not possible. This can't be done without sensing the size of the workflow. In this example, the assertion would ensure that the dataset contain no missing value, and will be expanded in one of the two different sub-plan after the sensing.

4.1.3 Planning with domain-dependent knowledge

The planning process generate a very wide set of plan. A way to restrict the number of generated plan is to use domain knowledge as constraint. Son et al. [7] describe two type of domain knowledge, temporal knowledge and procedural knowledge. Temporal knowledge is represented through the use of the temporal operators *until*, *always*, *eventually* and *next*, that allow to define temporal constraint. For example, we could ensure that the dataset is normalized once in the plan writing a constraint $always(normalize(inputdataset))$. Procedural knowledge can be seen as an under-specified sketch of the plans. An example of procedural knowledge is $a_1; (a_2|_3); a_4; f$. It represent plans which start with action a_1, followed by a_2 or a_3, followed by a_4, such that the formula f is true at the end of the plan. When a such knowledge is provided to a planner, it only needs to decide between action a_2 and a_3. Both of these domain knowledge provide more guidance to the planner, and thus improve it efficiency.

4.1.4 Preference-based planning

For our problem, i.e. creating data-mining workflows, it is not a difficult to generate valid plan. The challenge consist in finding high quality plan. It is thus essential to be able to build plan that not only achieve the specified goals, but that are also as conform as possible to the user preference over the plan properties. For this purpose [1] present a language designed to represent domain-specific, qualitative user preferences. Basically, the idea is to associate to formulas weights that correspond to the user preferences over the situations described by these formulas. It is worth to mention that the language proposed support the definition of temporally extended preferences, using the knowledge presented in Section 4.1.3.

The integration of all the techniques presented above brings the need for an approach of planning that has a great expressivity and allows to easily represent domain knowledge.

4.2 Answer Set Programming for planning

The framework chosen for this work is Answer Set Programming (ASP) [4]a modern logic programming language that is increasingly used to solve planning task due to its capacity to integrate domain knowledge in a simple, declarative way, and its adaptation to dynamic settings [ARSS15]. Answer Set Programming is a form of constraint declarative programming based on a non-monotonic logic. This non-monotonic approach, expressed through negation as failure, make ASP very suitable for knowledge representation, and for reasoning with incomplete knowledge which are point that particularly interest us.

4.2.1 ASP presentation

The idea of answer set programming is to design a program whose answer sets will correspond to solutions of the given problem. Then a solver is used to find a solution. We will use the solveur *clasp*.[2]

In its simplest form, an answer set program P on a set of atoms A consist in a set of rules of the form:

$$a_0 : -a_1, ..., a_m,\ not\ a_{m+1}, ...,\ not\ a_n.$$

where $0 \leq m \leq n$ and $a_i \in A$. The right part of the rule called *body* correspond to its premises, and the left part called *head* to its conclusion. The symbol *not* correspond to the negation as failure (NAF). Let P be a program that do not contain negation as failure. A set of atom X is said closed

[2]https://potassco.org/clasp/

under a positive program P (a program that do not contain negation as failure) for all rules $r \in P$, $head(r) \cap X = \emptyset$ whenever $body(r) \subseteq X$. X is an *answerset* for the program P if X is the minimal closed set of atom under P. An set of atom X is an answer set for a program P containing negation as failure if X is an answer set for its reduction P^X. The reduct of a program P with respect to a set of atom X is obtained by removing all rules from P containing $not\ a_i,\ a_i \in X$ and all NAF literals from the remaining rules.

4.2.2 Planning with ASP

ASP is very well suited for planning task, thus it is increasingly used. Many method for planning with ASP that can help us to design a planner for data-mining workflows have been developed.

V. Lifschitz [4] introduce the basics of ASP planning and present how the ramification problem (the problem of describing indirect effects of actions) is handled. With the action language BC, J. Lee [3] propose a solution for dealing with fluent whose default behavior is not inertia. The example proposed is that of a leaking container whose amount of which it contains automatically drops over time.

Son et al. [7] provide us solution to use domain-dependent knowledge with ASP planning. They consider temporal, procedural and HTN knowledge. For each kind of domain knowledge, they introduce new constructs for its encoding, and they present set of rules able to check if a constraint is violated.

The challenge will be to make good use of all those methods to propose a good modeling of our workflow generation problem.

5 Conclusion

KDD have been included in many everyday application which lead to a huge demand for data-miner. However, the KDD process is a complex task which can difficultly be achieved by novices. Their is thus a need for systems able to help data-miner during the KDD process. In this rapport, we have presented systems able to provide assistance during the data-mining workflow construction process and facilitate the work of data-miner. The system presented by Nguyen et al. [5] use both meta-mining and planning. It is a great progress in the direction we want to reach and provide us a baseline on which to work. In our idea, the system will need to interact with the user during the planning. The planning approach presented by Nguyen et al. [5] being to limited for this, we presented new planning methods than could be useful to achieve our idea of system. We also presented the programming language ASP which suit well to our needs.

The first challenge we will face is to use wisely the different planning methods to provide a good characterization of the different KDD operators to the ASP planner.

References

[1] M. Bienvenu, C. Fritz, and S. McIlraith. Specifying and computing preferred plans. 175(7-8):1308–1345, 2011.

[2] Sami Hanhijärvi, Kai Puolamäki, and Gemma C Garriga. Multiple hypothesis testing in pattern discovery. *arXiv preprint arXiv:0906.5263*, 2009.

[3] J. Lee, V. Lifschitz, and F. Yang. Action language BC: Preliminary report. pages 983–989.

[4] V. Lifschitz. Answer set programming and plan generation. 138(1-2):39–54, 2002.

[5] Phong Nguyen, Melanie Hilario, and Alexandros Kalousis. Using meta-mining to support data mining workflow planning and optimization. *Journal of Artificial Intelligence Research*, 51:605–644, 2014.

[6] Floarea Serban, Joaquin Vanschoren, Jörg-Uwe Kietz, and Abraham Bernstein. A survey of intelligent assistants for data analysis. *ACM Computing Surveys (CSUR)*, 45(3):31, 2013.

[7] T. Son, C. Baral, T. Nam, and S. McIlraith. Domain-dependent knowledge in answer set planning. 7(4):613–657, 2006.