7

Automating Workflow/Pipeline Design

Summary. This chapter discusses the design of *workflows* (or *pipelines*), which represent solutions that involve more than one algorithm. This is motivated by the fact that many tasks require such solutions. This problem is non-trivial, as the number of possible workflows (and their configurations) can be rather large. This chapter discusses various methods that can be used to restrict the design options and thus reduce the size of the configuration space. These include, for instance, ontologies and context-free grammars. Each of these formalisms has its merits and shortcomings. Many platforms have resorted to planning systems that use operators. These can be designed to be in accordance with the given ontologies or grammars. As the search space may be rather large, it is important to leverage prior experience. This topic is addressed in one of the sections, which discusses *rankings of plans* that have proved to be useful in the past. The workflows/pipelines that have proved successful in the past can be retrieved and used as plans in future tasks. Thus, it is possible to exploit both planning and metalearning.

7.1 Introduction

This chapter extends the discussion in Chapters 5 and 6, which discussed various approaches to selection of algorithms and their configurations. In many practical situations, the aim is to select not just one algorithm and its configuration, but rather a sequence of algorithms (or operators). An example of such a sequence could be to remove the outliers from the data, impute all missing values, and build a decision tree classifier on the dataset. Typically, such a sequence starts with one or several *data transformations* and ends with a machine learning algorithm (e.g., a classifier). Sometimes, post-processing operators are also defined. A typical example is a KDD task that may be resolved only by applying such a sequence.

The term *knowledge discovery from data* (KDD) was introduced at the first KDD workshop in 1989 (Piatetsky-Shapiro, 1991). This term emphasizes that "knowledge" is the

[1]The topic of this chapter was addressed already in the first edition, namely in Chapter 4 (Extending Metalearning to Data Mining and KDD (pp. 61–72)). This chapter was prepared by Christophe Giraud-Carrier, which we gratefully acknowledge. A part of this material was reused, revised, and reorganized. Besides, various new sections have been added.

P. Brazdil et al., *Metalearning*, Cognitive Technologies,
https://doi.org/10.1007/978-3-030-67024-5_7

end product of data-driven discovery. The reader may consult Figure 7.1 shown further on in this section. The term *data mining (DM)* is sometimes used as a synonym for KDD. Others consider it to be a part of the KDD process, focusing on the process of analyzing hidden patterns in the data according to different perspectives. Both terms, *KDD* and *DM*, have been popularized in AI and machine learning.

Designing sequences of operations has certain specificities, and these are addressed in this chapter. The sequences of algorithms are often referred to as *workflows* or *pipelines* of operations. In earlier work, some researchers called these sequences *DM processes* (Bernstein and Provost, 2001), others also *streams* (Engels et al., 1997; Wirth et al., 1997) or *plans*.

Many methods discussed in Chapters 5 and 6 are obviously also applicable to the more general task of designing workflows. However, we will not include the description of these methods here, simply to avoid duplication.

Chapter 14, which discusses the topic of automating data science (DS), addresses some specific problems that arise in the course of resolving typical DS tasks. So, the methods for the design of workflows (pipelines) may be reused in that setting too.

Organization of this chapter

When creating entire machine learning/KDD workflows (pipelines), the number of configuration options grows dramatically. It is therefore important to exclude the useless branches from the search space or to avoid them when conducting the search. Section 7.2 is dedicated to this topic. In this section we discuss how ontologies and grammars (e.g., CFGs) can be exploited for this aim. It is possible to use carefully designed abstract and concrete operators that follow the principles of ontologies or grammars. This topic is discussed in Section 7.3. In the same section we also discuss effective methods based on hierarchical planning that can search effectively through the given search space.

As the search space may be rather large, it is important to leverage prior experience. This topic is addressed in Section 7.4, which discusses rankings of plans that have proved to be useful in the past. Many of the techniques discussed in Chapter 2, where the focus was on rankings of algorithms, are also applicable to rankings of workflows.

Let us start by examining the KDD process, to provide a more precise idea about the kind of workflows we are concerned about in this chapter.

The KDD process

If we examine the KDD process in some detail (see Figure 7.1), it is clear that not all stages in it lend themselves naturally to automatic advice. Typically, both the early stages (e.g., problem formulation, domain understanding) and the late stages (e.g., interpretation, evaluation) require significant human input as they depend heavily on business knowledge.

The more algorithmic stages (i.e., preprocessing and model building), on the other hand, are ideal candidates for automation through adequate use of metaknowledge. Some decision systems focus exclusively on one of these stages, while others take a holistic approach, considering all stages of the KDD process collectively (i.e., as sequences of steps or workflows).

In this chapter we describe some basic concepts that have been followed in different data mining prototypes/systems that exploit metaknowledge to also provide decision support to users.

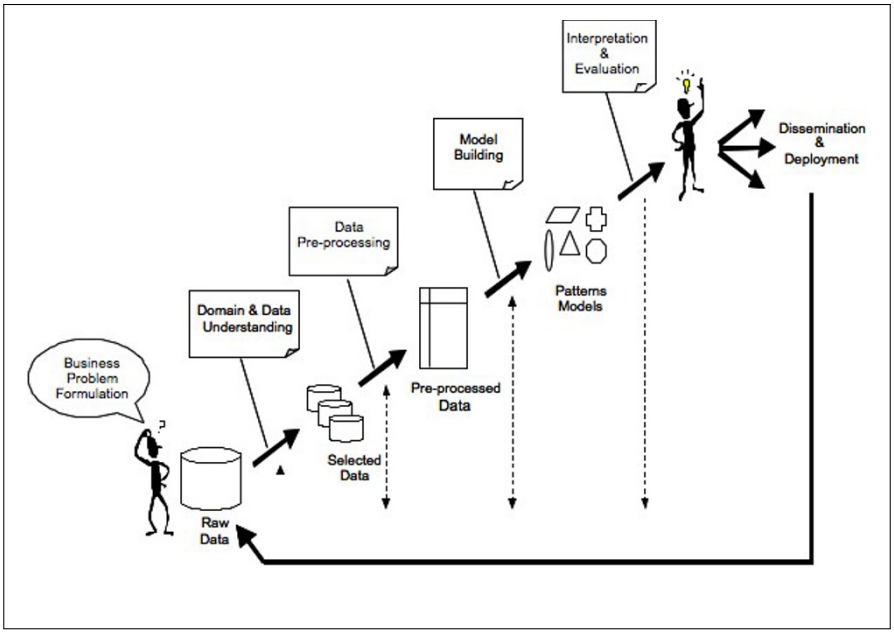


Fig. 7.1: The KDD process

7.2 Constraining the Search in Automatic Workflow Design

The designers of data mining systems need to address the following phases:

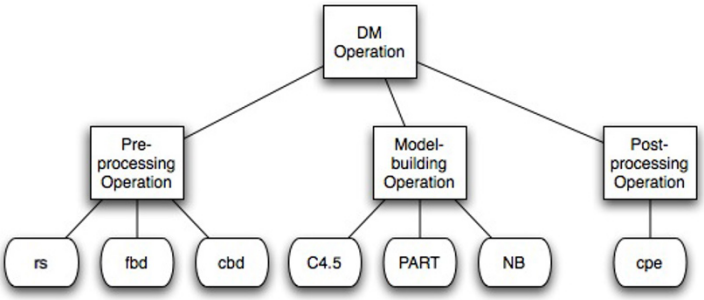
- Defining the space of alternative workflows (configuration space)
- Searching through the space of alternative workflows and selecting the most appropriate one(s) for the target dataset

Each of these is discussed in more detail in the following subsections.

7.2.1 Defining the space of alternatives (declarative bias)

The phase when the space of alternative workflows is defined normally precedes all the other stages. It takes into account the users' goals and determines which kind of metadata should be collected so that we could come up with a useful system. The definition of a space of alternatives is related to what some researchers in ML would call *declarative bias*. It specifies the representation of the space of hypotheses and affects the size of the search space (Mitchell, 1982; Gordon and desJardins, 1995). In the context of workflows, the definition of a space of alternatives involves two issues:

- Definition of the basic constituents (operators) of workflows
- Definition of the ways these constituents (operators) can be combined to generate workflows



rs = random sampling (10%), fbd = fixed-bin discretization (10 bins), cbd = class-based discretization, cpe = CPE-thresholding post-processor

Fig. 7.2: Sample IDA ontology

Some researchers have introduced a notion of *algorithm portfolios* to represent different sets/lists of algorithms/workflows (Gomes and Selmany, 2001; Leyton-Brown et al., 2003). These could be regarded as the basic constituents mentioned above. The disadvantage of this approach is that we cannot group similar algorithms into closely related groups. This problem can be overcome by adopting ontologies, which are discussed next.

Role of ontologies

Ontologies (Chandrasekaran and Jopheson, 1999) allow us to describe a set of constituents of workflows, often referred to as operators. They have been employed in various machine learning and data mining systems, including, for instance:

- DM ontology used in IDA (Bernstein and Provost, 2001)
- OWL-DL-based ontology used in the RDF system (Patel-Schneider et al., 2004)
- DMWF, Data Mining Workflow Ontology used in the eIDA system (Kietz et al., 2009)
- KDDONTO (Diamantini et al., 2012)
- DMOP (Hilario et al., 2009)
- KD ontology (Žáková et al., 2011)
- Exposé ontology (Vanschoren et al., 2012)
- Auto-Weka parameter space (Thornton et al., 2013)

More details about some of these ontologies can be found in an overview paper of Serban et al. (2013). Let us examine two ontologies in more detail.

Ontology used in the IDA A simplified ontology used in IDA system of Bernstein and Provost (2001) is shown in Figure 7.2. The bottom layer shows the *concrete operators*, such as *rs*, *fbd*, etc. The level above shows what some call *abstract operators*, which include *preprocessing*, *model-building* and *post-processing operation*.
An example of a workflow resulting from this ontology is "$fbd; nb; cpe$", meaning that first the data is discretized, afterwards a naive Bayes classifier is applied to it, and finally CPE-thresholding is applied to the predictions.

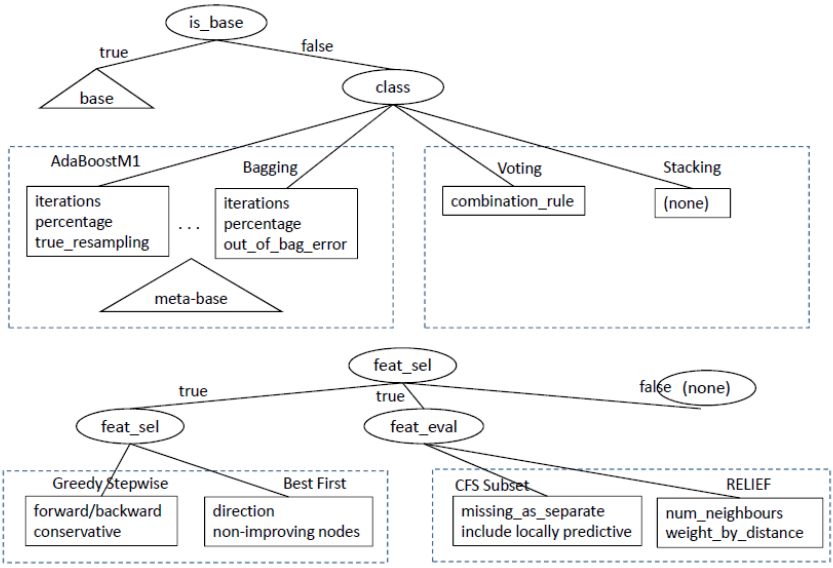


Fig. 7.3: Partial Auto-WEKA parameter space (redrawn following Thornton et al. (2013))

Ontology used in Auto-WEKA Figure 7.3 shows a part of the ontology used in Auto-WEKA (Thornton et al., 2013). The top part is concerned with the selection of classifications methods, which can be either *base classifiers* (left) or *ensemble methods* (right). The triangle with the word *base* inside represents a numeric parameter (index) determining which of the 27 base classifiers should be selected. Some components show the associated hyperparameters that need to be set. For example, the component "AdaBoostM1" requires the setting of three hyperparameters, namely "iterations", "percentage" and "true resampling". The bottom part of the figure is concerned with feature selection/evaluation methods.

What ontologies usually do not express

We note that ontologies typically do not say anything about the order in which the operations should be applied. We may be inclined to assume that the elements are searched through in left-to-right order, but this may not be so. Also, ontologies do not specify whether we should apply *all* or *just some* operators belonging to some branch.[1]

Note that ontologies can be used to generate workflows. However many components include multiple hyperparameters that need to be set. So search and optimization methods are required in this process (see Chapter 6).

Although the ontology helps us to construct all valid workflows, such as "$fbd; nb; cpe$", it permits to obtain various other workflows such as "$c4.5; nb$", that may not be consid-

[1]Although the language used to describe ontologies could, in principle, be refined to capture such aspects, the ontologies encountered do not do this.

ered valid. So we need other ways to determine the order in which the search space should be searched through. Apart from declarative bias, we also need to determine what some researchers call *procedural bias* (Mitchell, 1997). This issue is discussed in the next subsection.

7.2.2 Different ways of imposing procedural bias

Different mechanisms can be used to control the way a given space can be searched through. It can be done using:

- Heuristic rankers
- Context-free grammars
- Operators with preconditions and post-conditions (effects)

Using a heuristic ranker

The first possibility was explored in IDA (Bernstein and Provost, 2001). It uses a heuristic ranker which ranks the valid operations (processes) using a heuristic function which incorporates the user's preferences regarding speed, accuracy, and model comprehensibility. These characteristics are specified in the ontology for all operators. One problem with this approach is that these characteristics are acquired through experience on individual cases.

7.2.3 Context-free grammars (CFGs)

The formalism of context-free grammars (CFGs) enables to not only define the space to alternative models but also impose some restrictions regarding the search process, as we will see.

A CFG is a 4-tuple (S, V, Σ, R), where S is the starting symbol, V a set of non-terminal symbols, Σ a set of terminal symbols, and R a set of production rules specifying replacements of symbols (Hopcroft and Ullman, 1979).

The difference between the two types of symbols is that non-terminal symbols can be substituted by other symbols, while terminal symbols cannot. Valid substitutions are expressed in the form of production rules. The left-hand side of these rules includes a non-terminal symbol, while the right-hand side can include either a terminal symbol or a combination of non-terminals and terminals.

Example

Let us now see how we can use CFGs to represent the simple ontology shown in Figure 7.2. Let S represent the starting symbol. All non-terminals here are represented by strings starting with a capital letter. So, for instance, $Pre.Proc$ represents a single non-terminal corresponding to *Pre-Processing Operation* in Figure 7.2. So the set V would include this symbol, among others.

All terminal symbols are represented by strings in lower case. So, for instance, "c4.5" and "part" are terminals, representing particular classifiers. The set Σ would include these terminals among others.

Sequences of symbols are represented by the operator ";". The symbol "|" represents alternatives (logical OR). The null operation is represented by $null$. One possible set R of grammar rules is shown below:

1. $S \leftarrow DM.Oper$
2. $DM.Oper \leftarrow Pre.Proc\ ;\ Model.Build$
3. $Pre.Proc \leftarrow Sampling\ ;\ Discret$
4. $Sampling \leftarrow rs \mid null$
5. $Discret \leftarrow fbd \mid cbd \mid null$
6. $Model.Build \leftarrow Cat.Classif \mid Prob.Classif.Post$
7. $Cat.Classif \leftarrow c4.5 \mid part$
8. $Prob.Classif.Post \leftarrow nb\ ;\ cpe$

Let us see how some of the rules above can be interpreted. Line 1 states that the workflow should start with a "DM.Oper", which is defined on line 2. However, this is a non-terminal and hence needs to be substituted by other symbols. We note that this non-terminal corresponds to the abstract operator shown in the ontology in Fig. 7.2. Line 2 states that "DM.Oper" can be substituted by the sequence of "Pre.Proc" and "Model.Build", defined on line 3 and line 6, respectively.

The operation "Pre.Proc" (line 3), in turn, consists of a sequence of "Sampling" and "Discret", which are defined further on. Line 4 defines "Sampling" and introduces terminal symbols: either "rs" (representing random sampling) or "null' (null operation). These can be related to the concrete operators shown as *leaf nodes* in the ontology in Figure 7.2.

The grammar above enables to generate the workflows shown in Table 7.1, which correspond to a sample of DM processes generated by IDA. Note that the occurrences of "null" are typically not displayed.

Table 7.1: Sample of IDA-generated DM processes (workflows)

Nº	Workflow
1	c4.5
2	part
3	nb; cpe
4	rs; c4.5
5	rs; part
6	rs; nb; cpe
7	fbd; c4.5
8	fbd; part
9	fbd; nb; cpe
10	cbd; c4.5
11	cbd; part
12	cbd; nb; cpe
13	rs; fbd; c4.5
14	rs; fbd; part
15	rs; fbd; nb; cpe
16	rs; cbd; c4.5
17	rs; cbd; part
18	rs; cbd; nb; cpe

We note that the rules allow us to use the operation "cpe" only in conjunction with the probabilistic classifier "nb" that outputs probabilities. Also, it is not possible to obtain some invalid operator sequences, such as "$c4.5; nb$".[2] In other words, CFGs can capture certain aspects of procedural bias.

An alternative way of imposing procedural bias could be achieved by adopting the formalism of *context-sensitive grammars* (CSGs) (Martin, 2010; Linz, 2011).

Inducing CFGs from examples of workflows

Some users may find it easier to provide a list of valid/invalid workflows, rather than some formal procedure or grammar that could be executed. Various papers address the issue of grammar induction (see, e.g., Duda et al. (2001). The main idea is to spot frequent patterns, such as the sequence "$nb; cpe$" in the examples of workflows in Table 7.1. The frequent pattern identified can then be substituted by a new symbol, which could be renamed by the user to, say, *Prob.Classif.Post*. Besides, we also need to add a new rule showing how to interpret the new symbol. In our case we would get

$$Prob.Classif.Post \leftarrow nb; cpe.$$

In Chapter 15, we discuss various operations that enable to construct new concepts. These transformations could be used to introduce new higher-level concepts, corresponding to new non-terminals in CFGs and abstract operators discussed further on (see Subsection 7.3.4).

One problem with this approach is that, normally, many alternative grammars exist that are consistent with a given set of workflows. Another problem is that the new symbols introduced may not relate to the concepts of the user. Besides, we need a system that is capable of not only inducing a set of grammar rules but also revising them when new examples of workflows have been added.

Limitations of CFGs

CFGs have various limitations. First, it can happen that a particular operation OP_i can be transformed into a sequence OP_j in some contexts or into a sequence OP_k in others. This problem can be resolved by specifying the conditions under which each particular transformation should occur. This difficulty can be circumscribed in a formalism which exploits operators. This formalism is discussed in the following two sections.

7.3 Strategies Used in Workflow Design

Workflows can be constructed in different ways. It can be done manually either from scratch, or by modifying an existing workflow, or automatically with recourse to a planner. The first two options are discussed in the following subsections. The use of planning approaches is discussed in a separate section further on.

[2] To achieve that, the *model-building operations* were separated into *categorical classifiers*, which include "$c4.5, part$", and *probabilistic classifiers*, which include just "nb". The ontology could, of course, be redesigned to capture this aspect.

7.3.1 Operators

Operators were already used in the 1970s in the area of *automated planning (AI planning)*, which is a branch of AI (Russell and Norvig, 2016; Fikes and Nilsson, 1971). Its aim is to design strategies or action sequences that could be executed by some system. In classical planning, the system may be an intelligent agent or a robot. Here the aim of the planning system is to design workflows.

The selection of operators is guided by pre- and post-conditions, sometimes also called *effects*. Preconditions are conditions that must be met for the operation to be applicable. Post-conditions (effects) are conditions that are true after the operation is applied, i.e., how the operation changes the state.

7.3.2 Manual selection of operators

Manual construction is usually done nowadays with the aid of a palette, which includes all possible operators. The user then constructs a workflow by connecting together operations selected from the palette. The system can check preconditions, can make suggestions as to what operations might be required, and essentially maintains the integrity of the workflow. This approach was used, for instance, in CITRUS (Engels et al., 1997; Wirth et al., 1997).

7.3.3 Manual modification of existing workflows

If this alternative is taken, the user needs to provide a high-level description of the task at hand. The system acts as a kind of case-based reasoning system which searches for and identifies closest matches in past experiments. These may be real tasks previously performed or basic templates designed by experts. The closest match is presented to the user, who in turn can adapt it to the new target task.

This approach was used for instance in CITRUS (Engels et al., 1997; Wirth et al., 1997), discussed earlier, and in MiningMart. The latter was a large European research project which focused on algorithm selection for preprocessing rather than for model building (Euler et al., 2003; Euler and Scholz, 2004; Morik and Scholz, 2004; Euler, 2005).

Preprocessing generally consists of nontrivial sequences of operations or data transformations, and is widely recognized as the most time-consuming part of the KDD process, accounting for up to 80% of the overall effort. Hence, automatic guidance in this area can indeed greatly benefit users.

The goal of MiningMart is to enable the reuse of successful preprocessing phases across applications through case-based reasoning. A model for metadata, called M4, is used to capture information about both data and operator chains through a user-friendly computer interface. The complete description of a preprocessing phase in M4 makes up a *case*, which can be added to MiningMart's case base (MiningMartCB, 2003; Morik and Scholz, 2004):

"*To support the case designer a list of available operators and their overall categories, e.g., feature construction, clustering or sampling is part of the conceptual case model of M4. The idea is to offer a fixed set of powerful preprocessing operators, in order to offer a comfortable way of setting up cases on the one hand, and ensuring re-usability of cases on the other.*"

Given a new mining task, the user may search through MiningMart's case base for the case that seems most appropriate for the task at hand. M4 supports a kind of business level, at which connections between cases and business goals may be established. Its more informal descriptions are intended to "help decision makers to find a case tailored for their specific domain and problem."

A less ambitious attempt at assisting users with preprocessing has been proposed, with a specific focus on data transformation and feature construction (Phillips and Buchanan, 2001). The system works at the level of the set of attributes and their domains, and an ontology is used to transfer across tasks and suggest new attributes (in new tasks based on what was done in prior ones).

7.3.4 Using planning in workflow design

The classical definition of a planning problem is often formulated as follows. Given a description of the possible initial states of the world, a description of the desired goals, and a description of a set of possible actions (represented by operators), the aim of the planning system is to synthesize a plan that can generate a state which contains the desired goals.

We note that this definition does not quite apply to our problem and needs to be suitably adapted. For instance, when searching for a suitable workflow for a classification task, the aim is to achieve the highest possible accuracy, but we do not know a priori what value should be set as a goal.

The most commonly used languages for classical planning are STRIPS (Fikes and Nilsson, 1971) and PDDL (McDermott et al., 1998). These representations suffer from the curse of dimensionality, and some researchers have resorted to hierarchical planning, discussed further on. However, before that, we need to clarify what are abstract and concrete operators.

Abstract and base-level operators

Hierarchical planning enables to decompose complex tasks into less complex ones. Complex tasks are realized by *abstract operators* and primitive tasks by base-level operators, which in our case are base-level algorithms. These notions existed already in the early robot planning system STRIPS. Abstract operators were referred to as *intermediate-level actions* (ILAs) and concrete operators as *low-level actions* (LLAs) (Russell and Norvig, 2016).

A typical process involves mapping higher-level tasks to higher-level (abstract) operators, representing groups of operators at a lower level. The selection of operators is guided by pre- and post-conditions, sometimes also called *effects*.

Preconditions are conditions that must be met for the operation to be applicable (e.g., a discretization operation expects continuous inputs, a naive Bayes classifier works only with nominal inputs[3]). Post-conditions (effects) are conditions that are true after the operation is applied, i.e., how the operation changes the state of the data (e.g., all inputs are nominal following a discretization operation; a decision tree is produced by a decision tree learning algorithm). One obvious method is to define operators manually. However, this is quite a laborious task, particularly if the required set of operators is

[3]In some implementations, a discretization step is integrated, essentially allowing the naive Bayes classifier to act on any type of input.

large and if, in addition, many interactions exist. As the operators need to conform to given ontologies, some approaches exploit the ontological descriptions of operators to obtain the descriptions used in planning.

As the design of a set of operators could be quite a lengthy process, it would be useful to have a kind of support system that would enable to generate a set of operators capable of generating all valid workflows but none of the invalid ones.

How planning works

Many systems use a planner to design workflows. These include, for instance, CITRUS (Engels et al., 1997; Wirth et al., 1997), IDA (Bernstein and Provost, 2001), the eIDA system (Kietz et al., 2012), Auto-WEKA (Thornton et al., 2013; Kotthoff et al., 2016), and Auto-sklearn (Feurer et al., 2015), among others. A typical process involves the following steps.

The plan generator takes as input a dataset, a user-defined objective (e.g., build a classifier), and user-supplied information about the data, information that may not be obtained automatically. Starting with an empty process, it systematically searches for an operation whose preconditions are met and whose indicators are congruent with the user-defined preferences. Once an operation has been found, it is added to the current process and its post-conditions become the system's new conditions from which the search resumes. The search ends once a goal state has been reached or when it is clear that no satisfactory goal state may be reached.

In IDA the search of the planner is exhaustive: all valid workflows are returned. This was feasible at the time, as the problems studied did not involve large search spaces. However, even relatively simple problems may involve rather larger search spaces with hundreds or thousands of nodes, and so this approach is not really feasible. Hence we need to employ strategies that would help us to overcome these problems. These are discussed in the next subsection.

Exploiting hierarchical planning

Hierarchical planning goes back to the 1970s (Sacerdoti, 1974) and can be seen as a subarea of planning (Ghallab et al., 2004). As it exploits *hierarchical task networks* (HTN) (Kietz et al., 2009, 2012), it is sometimes referred to as HTN planning (Georgievski and Aiello, 2015). In some systems a reasoning engine is integrated in the planner directly by querying the ontology (Kietz et al., 2009; Žáková et al., 2011). Various other approaches are discussed by Serban et al. (2013).

The ML-Plan system (Mohr et al., 2018; Wever et al., 2018) is an AutoML system based on hierarchical planning. The authors have shown that it is highly competitive to some state-of-the-art systems, including Auto-WEKA, Auto-sklearn, and TPOT. Another planning system was discussed by Gil et al. (2018).

Tree-based pipeline optimization tool (TPOT)

This is a technique that uses evolutionary search to address the CASH problem (Olson et al., 2016). It works by evolving several small machine learning workflows into larger workflows. It starts by sampling a large amount of small pipelines, typically consisting of a data preprocessing operator and a single classifier. The *cross-over function* is defined by

merging the processed elements from two pipelines. Figure 7.4 illustrates this by means of the *Combine Features* operator. Only one of the classifiers prevails. When this process is repeated for several generations, pipelines can grow larger and more complex.

There are also various *mutation operators* which can adapt the pipeline by adding more operators or dropping some.

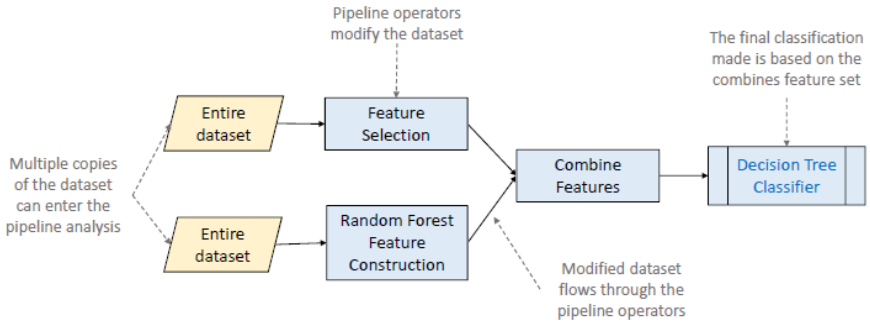


Fig. 7.4: An example of a tree-based pipeline, generated by TPOT (based on Olson et al. (2016))

Olson et al. (2016) focus on finding an appropriate pipeline with good preprocessing operations. The set of classifiers used in the experiments was restricted to decision trees and random forests.

General automatic machine learning assistant (GAMA)

The *general automatic machine learning assistant* (GAMA) also leverages a multi-objective genetic programming technique, similar to TPOT (Gijsbers and Vanschoren, 2019). However, while TPOT uses a synchronous strategy, evaluating all candidate pipelines in each generation before selecting the best one, GAMA uses an asynchronous approach, which is often faster since it is not slowed down by slow pipelines (stragglers) in a population. It is also much more modular: it allows the user to switch to other optimization techniques, e.g. asynchronous successive halving algorithm (ASHA), which is often faster on larger datasets, and allows post-processing, such as building an ensemble out of the pipelines evaluated during the search, similar to Auto-sklearn. It also includes logging and visualization of the search process to help researchers understand what it is doing.

Methods for reducing the search space

The main strategy for reducing the search space of planners relies on eliminating some parts of the search space, provided this does not affect the final outcome. This can be achieved in various ways. Previous strategies involved, for instance, prioritizing rules and also adding constraints (conditions) to rules to restrict their applicability (Brazdil, 1984; Clark and Niblett, 1989). These techniques can be reused in the design of operators.

Kietz et al. (2012) have pointed out that most of the preprocessing methods are recursive, handling one attribute at a time. It does not make sense to consider different orderings in which the attributes could be processed. If the dataset includes just two attributes, we could capture this by

$$O_p(At_1); O_p(At_2) \mid O_p(At_2); O_p(At_1), \tag{7.1}$$

where $O_p(At_j)$ represents the application of a particular preprocessing operator to attribute At_j. The formulation $O_p(At_1); O_p(At_2)$ restricts the search space and hence could be adopted, but does not quite express that all orderings lead to the same effect. The formulation in Eq. 7.1 is better, as it captures that the operations are commutative. The choice of the ordering that should be used is left to the interpreter.

If we were to reformulate this using operators with preconditions and effects, we would have to make sure that it is equivalent to what Eq. 7.1 expresses. In a more general case, which includes a set of attributes, simple enumeration of all alternatives, as is done in Eq. 7.1, is impractical, as for N items there are $N!$ orderings. To resolve this case we need higher-level constructs, which allow us to select one possible ordering from a set.

The system discussed by Kietz et al. (2012) goes part of the way in this direction. The operator *CleanMissingValues* is applied to all attributes, but the formalism does not capture the fact that one of the orderings was selected from a set.

Furthermore, the HTN planning method of Kietz et al. (2012) prevents senseless operator combinations. For example, first normalizing the data and then discretizing it does not make sense, as it produces a similar effect as applying just discretization. Converting the scalar data to nominal and then converting it back is useless. Eliminating such possibilities from the search space speeds up the process of planning.

Prioritizing the search

As planners can, in principle, generate a very large number of correct candidate workflows, other techniques need to be applied to control the search. Different possibilities can be exploited:

- Ask the planner to return a different subset of workflows (limited number) each time it is invoked;
- Exploit meta-learning to identify and rank the potentially best workflows. Various chapters (e.g., 2 and 5) in this book provide more details about this process;
- Exploit sequential model-based optimization (SMBO) to suggest the potentially best workflow(s) to test. Chapter 6 discusses this issue in more detail.

The reader can consult the respective chapters to obtain more information about each of these techniques.

Exploiting metaknowledge in planning

Nguyen et al. (2014) exploit a meta-mining model which is used to guide the planner during the workflow planning. The meta-mining models are learned using a similarity learning approach. These models extract workflow descriptors by mining the workflows for generalized relational patterns. This mechanism allows us to focus the search on components recommended by the meta-learner.

Methods for revision workflows (pipelines)

AlphaD3M (Drori et al., 2018) is an AutoML system that uses meta reinforcement learning on sequence models generated by self play. The current state is represented by the current pipeline, and possible actions include addition, deletion, or replacement of pipeline components. A Monte Carlo tree search (MCTS) (Silver et al., 2017; Anthony et al., 2017) generates pipelines, which are evaluated to train a recurrent neural network (LSTM) that can predict pipeline performance, in turn producing the action probabilities for the MCTS in the next round. The state description also includes metafeatures of the current task, allowing the neural network to learn across tasks. AlphaD3M was compared with state-of-the-art AutoML systems: Auto-sklearn, Autostacker, and TPOT on OpenML datasets. The authors claim that this system achieves competitive performance while being an order of magnitude faster.

Alternatively, Naive AutoML is a baseline for pipeline search, where each component is searched based on an independence assumption (Mohr and Wever, 2021). Each component is optimized independent from other components, shrinking the search space.

7.4 Exploiting Rankings of Successful Plans (Workflows)

This chapter extends the material presented in Chapter 2 which discussed rankings of algorithms. Here, we consider rankings of workflows rather than just individual algorithms. The basic idea consists of storing all potentially useful workflows that were identified in the past for future use. Normally the workflows are ranked according to a given estimate of how useful they were in the past, for instance, *average rank* across different datasets. The assumption here is that the ranking can help us to identify the potentially best workflow on the new dataset.

Serban et al. (2013) refer to these kinds of approaches as *case-based reasoning* (CBR) approaches. The idea of storing ranked cases is not new. The idea was explored already in the 1990s in the Statlog and Metal projects and in the subsequent design of Data Mining Advisor (DMA) (Brazdil and Henery, 1994; Giraud-Carrier, 2005) and AST (Lindner and Studer, 1999). These systems stored algorithms rather than workflows. Other systems, including, for instance, CITRUS (Engels et al., 1997; Wirth et al., 1997) or MiningMart (Euler et al., 2003; Euler and Scholz, 2004; Morik and Scholz, 2004; Euler, 2005), stored workflows.

The idea of storing more complex constructs which are potentially useful in future tasks has appeared in different forms in the past. The early planning system STRIPS introduced the possibility of storing the results of planning in the form of generalized macro-operators (Russell and Norvig, 2016; Fikes and Nilsson, 1971).

Tabling, also referred to as *tabulation* or *memoing*, was proposed already in late 1960s (Michie, 1968). It consists of storing intermediate results for subgoals so that they can be reused later when the same subgoal appears again. In the area of logic programming, tabling is a form of automatic caching or memorization of results of previous computations. Storing them can avoid unnecessary recomputation.

Effectiveness of this approach

It was shown that a ranked case base of workflows can achieve better results than Auto-WEKA. Cachada et al. (2017) have exploited the AR* metalearning method in the com-

Table 7.2: Comparing AR* with workflows and Auto-WEKA

Budget	Win	Loss	Tie
5	35	1	1
15	31	5	1
30	29	7	1
60	7	11	1

parisons. The advantage of the ranked case base approach was particularly significant when the given time budgets were small. More details about this study follow.

In one of the experiments the portfolio of workflows included 184 elements, corresponding to 62 algorithms with default configurations, 30 variants with varied hyperparameter configurations (3 versions of MLP, 7 of SVM, 7 of RFs, 8 of J48, and 5 of k-NN) and the same number of variants (62+30), which were created by also including feature selection (CFS method (Hall, 1999)). One hundred datasets were used from the OpenML benchmark suite (Vanschoren et al., 2014) in a leave-one-out mode in the evaluation.

Auto-WEKA was run and only the first recommendation was used for each dataset. The Auto-WEKA total runtime was computed by adding the search time to the recommended model runtime. This runtime was used to retrieve the actual performance of the AR* system. The results for four different time budgets (in minutes) are shown in Table 7.2.

Portfolios of successful workflows

The concept of a *portfolio of workflows* can be seen as generalizations of the concept of a *portfolio of algorithms*. So a question arises as to which workflows should be kept for future use.

In Chapter 8 we address the issue of how to set up configuration spaces and portfolios. Interested readers can consult this chapter to find answers to this issue.

References

Anthony, T., Tian, Z., and Barber, D. (2017). Thinking fast and slow with deep learning and tree search. In *Conference on Neural Information Processing Systems*.

Bernstein, A. and Provost, F. (2001). An intelligent assistant for the knowledge discovery process. In Hsu, W., Kargupta, H., Liu, H., and Street, N., editors, *Proceedings of the IJCAI-01 Workshop on Wrappers for Performance Enhancement in KDD*.

Brazdil, P. (1984). Use of derivation trees in discrimination. In O'Shea, T., editor, *ECAI 1984 - Proceedings of 6th European Conference on Artificial Intelligence*, pages 239–244. North-Holland.

Brazdil, P. and Henery, R. J. (1994). Analysis of results. In Michie, D., Spiegelhalter, D. J., and Taylor, C. C., editors, *Machine Learning, Neural and Statistical Classification*, chapter 10, pages 175–212. Ellis Horwood.

Cachada, M., Abdulrahman, S., and Brazdil, P. (2017). Combining feature and algorithm hyperparameter selection using some metalearning methods. In *Proc. of Workshop AutoML 2017, CEUR Proceedings Vol-1998*, pages 75–87.

Chandrasekaran, B. and Jopheson, J. (1999). What are ontologies, and why do we need them? *IEEE Intelligent Systems*, 14(1):20–26.

Clark, P. and Niblett, T. (1989). The CN2 induction algorithm. *Machine Learning*, 3(4):261–283.

Diamantini, C., Potena, D., and Storti, E. (2012). KDDONTO: An ontology for discovery and composition of KDD algorithms. In *Proceedings of the ECML-PKDD'09 Workshop on Service-oriented Knowledge Discovery*, pages 13–24.

Drori, I., Krishnamurthy, Y., Rampin, R., de Paula Lourenco, R., Ono, J. P., Cho, K., Silva, C., and Freire, J. (2018). AlphaD3M: Machine learning pipeline synthesis. In *Workshop AutoML 2018 @ ICML/IJCAI-ECAI*. Available at site https://sites.google.com/site/automl2018icml/accepted-papers.

Duda, R. O., Hart, P. E., and Stork, D. G. (2001). *Pattern Classification (2 ed.)*. John Wiley & Sons, New York.

Engels, R., Lindner, G., and Studer, R. (1997). A guided tour through the data mining jungle. In *Proceedings of the Third International Conference on Knowledge Discovery and Data Mining*, pages 163–166. AAAI.

Euler, T. (2005). Publishing operational models of data mining case studies. In *Proceedings of the ICDM Workshop on Data Mining Case Studies*, pages 99–106.

Euler, T., Morik, K., and Scholz, M. (2003). MiningMart: Sharing successful KDD processes. In *LLWA 2003 – Tagungsband der GI-Workshop-Woche Lehren–Lernen–Wissen–Adaptivitat*, pages 121–122.

Euler, T. and Scholz, M. (2004). Using ontologies in a KDD workbench. In *Proceedings of the ECML/PKDD Workshop on Knowledge Discovery and Ontologies*, pages 103–108.

Feurer, M., Klein, A., Eggensperger, K., Springenberg, J., Blum, M., and Hutter, F. (2015). Efficient and robust automated machine learning. In Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, NIPS'15, pages 2962–2970. Curran Associates, Inc.

Fikes, R. E. and Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2(3–4):189–208.

Georgievski, I. and Aiello, M. (2015). HTN planning: Overview, comparison, and beyond. *Artificial Intelligence*, 222:124–156.

Ghallab, M., Nau, D. S., and Traverso, P. (2004). *Automated planning - theory and practice*. Elsevier.

Gijsbers, P. and Vanschoren, J. (2019). GAMA: Genetic automated machine learning assistant. *Journal of Open Source Software*, 4(33):1132.

Gil, Y., Yao, K.-T., Ratnakar, V., Garijo, D., Steeg, G. V., Szekely, P., Brekelmans, R., Kejriwal, M., Luo, F., and Huang, I.-H. (2018). P4ML: A phased performance-based pipeline planner for automated machine learning. In *Workshop AutoML 2018 @ ICML/IJCAI-ECAI*. Available at site https://sites.google.com/site/automl2018icml/accepted-papers.

Giraud-Carrier, C. (2005). The Data Mining Advisor: Meta-learning at the Service of Practitioners. In *Proceedings of the International Conference on Machine Learning and Applications (ICMLA)*, page 113–119.

Gomes, C. P. and Selmany, B. (2001). Algorithm portfolios. *Artificial Intelligence*, 126(1-2):43–62.

Gordon, D. and desJardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1/2):5–22.

Hall, M. (1999). *Correlation-based feature selection for machine learning*. PhD thesis, University of Waikato.

Hilario, M., Kalousis, A., Nguyen, P., and Woznica, A. (2009). A data mining ontology for algorithm selection and meta-mining. In *Proceedings of the ECML-PKDD'09 Workshop on Service-Oriented Knowledge Discovery*, page 76–87.

Hopcroft, J. E. and Ullman, J. D. (1979). *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley.

Kietz, J., Serban, F., Bernstein, A., and Fisher, S. (2009). Towards cooperative planning of data mining workflows. In *Proceedings of ECML-PKDD'09 Workshop on Service Oriented Knowledge Discovery*, pages 1–12.

Kietz, J.-U., Serban, F., Bernstein, A., and Fischer, S. (2012). Designing KDD-Workflows via HTN-Planning for Intelligent Discovery Assistance. In Vanschoren, J., Brazdil, P., and Kietz, J.-U., editors, *PlanLearn-2012, 5th Planning to Learn Workshop WS28 at ECAI-2012, Montpellier, France*.

Kotthoff, L., Thornton, C., Hoos, H. H., Hutter, F., and Leyton-Brown, K. (2016). Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 17:1–5.

Leyton-Brown, K., Nudelman, E., Andrew, G., McFadden, J., and Shoham, Y. (2003). A portfolio approach to algorithm selection. In *International Joint Conferences on Artificial Intelligence (IJCAI)*, pages 1542–1543.

Lindner, G. and Studer, R. (1999). AST: Support for algorithm selection with a CBR approach. In Giraud-Carrier, C. and Pfahringer, B., editors, *Recent Advances in Meta-Learning and Future Work*, pages 38–47. J. Stefan Institute.

Linz, P. (2011). *An Introduction to Formal Languages and Automata*. Jones & Bartlett Publishers.

Martin, J. C. (2010). *Introduction to Languages and the Theory of Computation (4th ed.)*. McGraw-Hill.

McDermott, D., Ghallab, M., Howe, A., Knoblock, C., Ram, A., Veloso, M., Weld, D., and Wilkins, D. (1998). PDDL—the planning domain definition language. Technical report, New Haven, CT: Yale Center for Computational Vision and Control.

Michie, D. (1968). Memo functions and machine learning. *Nature*, 2018:19–22.

MiningMartCB (2003). MiningMart Internet case base. http://mmart.cs.uni-dortmund.de/end-user/caseBase.html.

Mitchell, T. (1982). Generalization as Search. *Artificial Intelligence*, 18(2):203–226.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill.

Mohr, F. and Wever, M. (2021). Naive automated machine learning–a late baseline for automl. *arXiv preprint arXiv:2103.10496*.

Mohr, F., Wever, M., and Hüllermeier, E. (2018). ML-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515.

Morik, K. and Scholz, M. (2004). The MiningMart approach to knowledge discovery in databases. In Zhong, N. and Liu, J., editors, *Intelligent Technologies for Information Analysis*, chapter 3, pages 47–65. Springer. Available from `http://www-ai.cs.uni-dortmund.de/MMWEB`.

Nguyen, P., Hilario, M., and Kalousis, A. (2014). Using meta-mining to support data mining workflow planning and optimization. *Journal of Artificial Intelligence Research*, 51:605–644.

Olson, R. S., Bartley, N., Urbanowicz, R. J., and Moore, J. H. (2016). Evaluation of a tree-based pipeline optimization tool for automating data science. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, pages 485–492.

Patel-Schneider, P., Hayes, P., and Horrocks, I. e. a. (2004). OWL web ontology language semantics and abstract syntax. W3C recommendation 10.

Phillips, J. and Buchanan, B. G. (2001). Ontology-guided knowledge discovery in databases. In *Proceedings of the First International Conference on Knowledge Capture*, pages 123–130.

Piatetsky-Shapiro, G. (1991). Knowledge discovery in real databases. *AI Magazine*.

Russell, S. J. and Norvig, P. (2016). *Artificial Intelligence: A Modern Approach*. Prentice Hall, 3rd edition.

Sacerdoti, E. (1974). Planning in a hierarchy of abstraction spaces. *Artificial Intelligence*, 5(2):115–135.

Serban, F., Vanschoren, J., Kietz, J., and Bernstein, A. (2013). A survey of intelligent assistants for data analysis. *ACM Comput. Surv.*, 45(3):1–35.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., and et al., T. G. (2017). Mastering chess and shogi by self-play with a general reinforcement learning algorithm. In *Conference on Neural Information Processing Systems*.

Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 847–855. ACM.

Vanschoren, J., Blockeel, H., Pfahringer, B., and Holmes, G. (2012). Experiment databases: a new way to share, organize and learn from experiments. *Machine Learning*, 87(2):127–158.

Vanschoren, J., van Rijn, J. N., Bischl, B., and Torgo, L. (2014). OpenML: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60.

Wever, M., Mohr, F., and Hüllermeier, E. (2018). ML-plan for unlimited-length machine learning pipelines. In *AutoML Workshop at ICML-2018*.

Wirth, R., Shearer, C., Grimmer, U., Reinartz, T. P., Schlosser, J., Breitner, C., Engels, R., and Lindner, G. (1997). Towards process-oriented tool support for knowledge discovery in databases. In *Proceedings of the First European Conference on Principles and Practice of Knowledge Discovery in Databases*, pages 243–253.

Žáková, M., Křemen, P., Železný, F., and Lavrač, N. (2011). Automating knowledge discovery workflow composition through ontology-based planning. *IEEE Transactions on Automation Science and Engineering*, 8:253–264.