2024-11-20

Ontology-Enhanced Automated Machine Learning

Davies, Cooper T.S.

UNIVERSITY OF CALGARY

Ontology-Enhanced Automated Machine Learning

by

Cooper T.S. Davies

A THESIS
SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA
NOVEMBER, 2024

Abstract

This thesis addresses the challenge of bridging the gap between traditional Problem-Specific Machine Learning (PSML) and Automated Machine Learning (AutoML) systems. While PSML offers high accuracy but demands substantial expertise, AutoML aims to automate the process of building a machine learning (ML) model but often lacks domain-specific knowledge. To address this, we propose Ontology-Enhanced AutoML, a novel approach that integrates domain knowledge from ontologies into the AutoML pipeline.

We first examine the current landscape of AutoML, highlighting the complexities faced by a system in selecting appropriate algorithms and hyperparameters. We identify the limitations of existing AutoML systems, particularly their blind reliance on datasets, which often leads to poor performance and lengthy training times.

Our thesis presents experiments demonstrating the effectiveness of Ontology-Enhanced AutoML in mitigating these challenges. By incorporating mechanisms for ontology-based feature extraction and example filtering, we demonstrate significant improvements in accuracy and optimization time compared to traditional AutoML. These results highlight the potential of Ontology-Enhanced AutoML to provide a wide range of systems lying between the extremes of PSML and AutoML.

This thesis contributes not only a technical solution but also a conceptual framework for understanding ML as a spectrum. We discuss implications for future research and the potential for further advancements in bridging the gap between domain expertise and ML proficiency.

Acknowledgements

I would like to express my utmost appreciation, gratitude, and thanks to the following individuals. Without them this document could never exist.

Firstly, I am forever grateful for the guidance and support of Dr. Jörg Denzinger. His knowledge on Machine Learning is as vast as it is insightful. Most of what I know regarding Machine Learning and Artificial Intelligence is due to Dr. Denzinger and it has been a pleasure working on this thesis under his supervision. I will forever be grateful for the time and effort he has contributed towards my studies. Dr. Denzginer has been instrumental in my growth during my time at the University of Calgary.

I would also like to acknowledge Dr. Frank Maurer. By taking a chance on a student with sub-par performance in classes Dr. Maurer is the man who started my path of graduate studies to begin with. Without Dr. Maurer my life would be completely different from where it is today. Because of this, I cannot thank Dr. Maurer enough.

I am also thankful for the members of my committee and defense, Dr. Robert Walker, Dr. Jeffrey Boyd, Dr. Christian Jacob, Dr. Scott Dick, and the neutral chair Dr. John Aycock. The amount of effort put into reading, reviewing, and proctoring by these individuals is substantial and deserving of recognition. The feedback provided by these individuals has also been invaluable in helping to produce this thesis.

Lastly, I am incredibly grateful to my loving wife, Lisa Davies. Her encouragement, support, and patience means the world to me. She has provided me with unending motivation that is much needed when achieving a goal as large as a Ph.D. I thank her for the sacrifices

she has made in my pursuit. Words cannot express my love for her.

Table of Contents

List of Figures

List of Tables

List of Symbols, Abbreviations, and Nomenclature

Abbreviation	Definition
AutoCV	Automated Computer Vision
AutoDL	Automated Deep Learning
AutoML	Automated Machine Learning
AutoNLP	Automated Natural Language Processing
AutoSpeech	Automated Speech Recognition
B-REPS	Boundary Representation
BOHB	Bayesian Optimization with Hyperband
CAD	Computer Aided Design
CASH	The Combined Algorithm Selection and Hyperparameter Optimization Problem
CNN	Convolutional Neural Network
FEA	Finite Element Analysis
GA	Genetic Algorithm
IDA	Independent Component Analysis
LDA	Linear Discriminant Analysis
ML	Machine Learning
NAS	Neural Architecture Search
NFL	No Free Lunch Theorem
NLP	Natural Language Processing
OWL	Web Ontology Language
PCA	Principal Component Analysis
PSML	Problem-Specific Machine Learning
QDA	Quadratic Discriminant Analysis
RDF	Resource Description Framework
RNN	Recurrent Neural Network
SMAC	Sequential Model-Based Optimization for General Algorithm Configuration
SVD	Singular Value Decomposition
SVM	Support Vector Machine
SWRL	Semantic Web Reasoning Language
TPE	Tree-of-Parzen Estimators

Symbol	Definition
$\mathcal{D}$	The set of all datasets
D	A dataset
$\mathcal{A}$	The set of all algorithms
A	An algorithm
Λ	The set of associated hyperparameters for an algorithm
${A_\lambda}^{(i,j)}$	Algorithm A_i with hyperparameters λ_j
$\mathcal{M}({A_\lambda}^{(i,j)}, D_{train}, D_{test})$	The misclassification of some algorithm with associated hyperparameters on a dataset
$\Phi(D)$	The set of all possible algorithms with associated hyperparameters, and their misclassification rate when applied to D
C	A classifier
$\mathcal{H}$	The set of all classifiers (hypothesis set)
$\mathcal{L}$	A learner

Chapter 1

Introduction

In recent times, Machine Learning (ML) has become quite popular with the general public after feats performed by systems such as AlphaFold [Jum+21], AlphaZero [Sil+18], and ChatGPT [AI23]. For example, people commonly utilize ML to automate tasks such as image classification [Rus+15], or to learn previously unknown information found within a domain [KT11]. Typically, a business will employ an ML specialist to build and configure an ML system to produce classifiers[i] that could be used to solve desired problems. The production of an accurate classifier is a complex task that requires intimate knowledge of both the domain of application of the encompassing problem, and extensive knowledge of ML. Due to the large variety and sophistication of learning algorithms, the barrier of entry for a non-ML expert is high. Even with open-source systems such as WEKA [Hal+09], Scikit-learn [Ped+11], and Accord.Net [Sou14], customizing ML systems is often vast and daunting, albeit required to learn a successful classifier.

The lack of an ML expert often leads domain experts to simply select learning algorithms that are popularized in blogs or articles, as opposed to ones that would be better suited for the given task [SLA12]. Conversely, the lack of a domain expert potentially leads to an ML expert instantiating an ML system that produces a classifier that has learned superficial or

[i]Within ML it is common to refer to the produced object as a model. However, as we discuss in future chapters, a model is more broad than a classifier. Within this thesis, the typical learning tasks are classification and thus the produced models are classifiers.

redundant patterns. Requiring both an ML expert and a domain expert makes instantiating an ML system logistically difficult and expensive. The ML expert and the domain expert often need to become somewhat of an expert in each other's field to accurately discuss, diagnose, and solve the underlying problem of instantiating an appropriate ML system.

The field of Automated Machine Learning (AutoML) has grown from the need of individuals wanting to use ML to produce classifiers without becoming ML experts themselves [Feu+15][KJC16][HKV19]. A significant problem with the traditional approach to ML is the vast number of learning algorithms an ML expert can choose from. This problem is compounded by the fact that each learning algorithm also requires configuring the associated hyperparameters which must be decided before learning occurs [NLA24]. Selecting the correct learning algorithm, as well as the correct hyperparameter configuration is a non-trivial task that requires in-depth knowledge of ML. Automating this process requires identifying a method for determining both a suitable learning algorithm, as well as the hyperparameter configuration for said learning algorithm.

However, a downside of AutoML systems is the broad nature of their uses [NP19][NLA24]. This leads to AutoML systems taking longer compute time to produce a classifier than typical ML systems as they must explore and infer patterns from scratch without the benefit of pre-encoded domain expertise. Another drawback is that an AutoML system relies solely on a single given dataset to learn from [NP19]. In essence, an AutoML system is blind to any information not explicitly represented in a dataset. This means that all domain knowledge that is not explicitly present in the dataset has to be learned at runtime. Consequently, AutoML systems often require more computational resources and time to achieve comparable performance, as they must perform extensive search and optimization to discover effective models and features. This trade-off between generality and efficiency is a key consideration when evaluating the practicality and performance of AutoML solutions [KKK22][ZH21].

Creators of ML systems would normally incorporate knowledge into a more specialized ML system during development whereas knowledge has to be learned at runtime by AutoML

systems. Additionally, knowledge that may be common sense in certain domains may often not be explicitly represented in a dataset. This results in AutoML systems producing classifiers that have potentially learned superficial patterns, resulting in superficial or poor performance.

An ideal AutoML system would work as fast as a domain expert desires, while also being able to produce results (classifiers) that have perfect accuracy. In the real world, however, the no free lunch theorem [WM95][WM97] and computational complexity limit the capabilities of any optimization function. This, in turn, force AutoML systems to have various shortcomings. These shortcomings are formalized as a *budget* that a system must adhere to. The budget of an AutoML system is typically described as some ratio between being as fast as possible to produce a classifier and having the produced classifier be as accurate as possible after being learned. Without knowing ahead of time what types of datasets an AutoML system will encounter, any AutoML system is likely going to have a low ceiling on efficiency.

The 2017/2018 ChaLearn AutoML competition [Guy+19] provides evidence of this where winning submissions achieve relatively inaccurate and inconsistent results (accuracies ranging from 0.28 to 0.69 depending on the system and the dataset). Additional evidence suggesting that experts consider AutoML ineffective is the fact that 2017/2018 was the last year ChaLearn hosted an AutoML competition. Compare this to the relatively high accuracy requirements of learned classifiers considered acceptable in academic settings, and the limitations of AutoML become apparent [BSI19][KSH12]. Often, Problem-Specific Machine Learning (PSML) systems will substantially out-compete AutoML in budget.

We view the distinction between AutoML and PSML as ends of a spectrum rather than as separate entities. Because of this view, we see the advantages and disadvantages of both AutoML and PSML as also existing on a spectrum. By viewing ML as a spectrum between AutoML and PSML we hypothesize that developers can create ML systems that allow users to systematically choose a system based on what level of generality, and conversely, what

level of accuracy they require. With this approach, a user of such an ML system would be able to determine how involved an ML expert would need to be for their use case. The level of involvement ranges from no involvement, in the case of AutoML, to fully hands-on, in the case of PSML. Choosing which ML system to use based on its placement on the spectrum between AutoML and PSML would give individuals the power to utilize ML in accordance with their knowledge of ML.

The ability to selectively choose an ML system with a specific combination of accuracy and generality enables individuals with no prior knowledge of ML to utilize ML to their benefit. For example, a researcher in astronomy wishing to create a classifier for predicting specific measurements of galaxies would not need to learn about ML concepts. Instead, they could select an ML system closer to AutoML than PSML, which would allow them to incorporate knowledge from their domain in a predefined manner. This choice might result in a system that performs less well than a PSML system but, with knowledge from their domain included, would perform better than AutoML systems. In contrast, individuals who want to produce highly accurate classifiers often have to first educate themselves on concepts such as learning rate, vanishing gradients, similarity functions, etc. Conversely, by using traditional AutoML they may end up with a classifier that is too inaccurate or slow to create.

To illustrate the spectrum of ML, we have designed methods that show how to specialize an AutoML system to include some domain knowledge. Our first method uses a mechanism to understand an ontology for extracting domain knowledge from an engineered ontology to perform feature extraction. We then developed a second method that employs a mechanism to understand an ontology and filter out examples from a dataset that contain already known knowledge. By filtering out examples with known knowledge, the AutoML system can focus on learning unknown knowledge. Incorporating these mechanisms into an AutoML system allows us to highlight a middle ground between PSML and AutoML on the spectrum; an area we call Ontology-Enhanced AutoML.

The abstraction of ML from the user's perspective means that users do not need to be experts in ML to benefit from it. Individual Ontology-Enhanced AutoML systems can specify the format that domain knowledge should follow within an ontology, allowing domain experts to create ontologies for these systems. Once experts engineer an ontology, they can share it with other users, further abstracting the need for deep domain knowledge. As a result, users can selectively choose which Ontology-Enhanced system to use and which ontology best suits their dataset, greatly reducing the need for users to do more than simply input their dataset into the system.

Our experimental results demonstrate the effectiveness of our attempt to bridge the gap between PSML and AutoML. We created an Ontology-Enhanced AutoML system that outperformed an AutoML system on two different datasets from similar domains using the same ontology. When we used only the mechanism for feature extraction from an ontology, we observed an increase in accuracy of 10% from 0.89 to 0.99 for one dataset and an increase of 13% from 0.87 to 1.0 for the other dataset. When we also implemented the mechanism for filtering out examples from a dataset, we further reduced optimization time by 63% from 1326 seconds to 485 seconds for one dataset and by 93% from 1668 seconds to 105 seconds for the other dataset.

Our experimental results demonstrate that we can view ML as a spectrum and create ML systems that occupy different points along that spectrum. By incorporating the feature extraction mechanism for ontological reasoning into an Ontology-Enhanced AutoML system, we show that the system can move a discrete "step" closer to PSML on the spectrum. When we add the ontological reasoning mechanism that filters out examples from a dataset, we further move the Ontology-Enhanced AutoML system closer to PSML. This demonstrates that we can develop an ML system to occupy a specific region on the ML spectrum using Ontology-Enhanced AutoML. Additionally, by applying the same ontology across different datasets, we show that we can take "steps" on the ML spectrum by only modifying the ML system. Users can supply domain knowledge through an ontology of their choice. From the

user's perspective, the key difference lies in which system they select for their use case.

In the following chapter, we will define the basic concepts and provide the definitions necessary to understand the scope of our contribution. Next, in Chapter 3, we will present our main concept. After introducing our concept, we will showcase our experimental results and discuss them in Chapter 4. We will then review related work in Chapter 5. Finally, in Chapter 6, we will conclude this thesis and suggest possible avenues for future research.

Chapter 2

Basic Definitions and Concepts

This chapter provides the reader with background definitions and concepts required to understand the contribution of this thesis. To begin we will discuss AutoML in Section 2.1. We will then discuss datasets in Section 2.2. Following datasets we cover ontologies in Section 2.3. After ontologies, we discuss computational geometry in Section 2.4. Lastly, Section 2.5. highlights the differences between the various approaches to ML.

2.1 AutoML

At a high level, an AutoML system acts as a function that can take in a dataset, and for it, create instantiations of ML systems that are used to learn a model from the dataset. AutoML achieves this by attempting to instantiate ML systems that best suit a learning task given a dataset. Each instantiated ML system then uses the dataset to create a model that can be queried to predict the class label for a given example. Each ML system instantiation creates individual models, that are then combined in an ensemble to produce a classifier. An AutoML system then provides the classifier as output.

AutoML systems utilize a pipeline of components that individually perform specific actions. These components, in conjunction with one another, can create multiple ML system instantiations from a given dataset and, for each instantiation, create models. An AutoML

system then measures the performance of the models produced by each ML instantiation. Using the results of each ML instantiation, an AutoML system selectively chooses better ML systems to further instantiate iteratively. Based on the results of the models created by each ML system instantiation created, an AutoML system iteratively refines its understanding of which ML systems will likely yield better results during runtime. After iterating to the point where improvements in accuracy are not as frequently recorded, the AutoML system can combine all the models created during these iterations in an ensemble to produce a classifier that contains all the knowledge it learned from a dataset during runtime.

To instantiate an ML system, an AutoML system must first select a learner to use within the ML system. Thus we first cover a section on the definition of the Algorithm Selection problem [Ric76]. A second problem is selection of the appropriate values for the hyperparameters of the chosen learner.

The term "hyperparameter" is commonly used to indicate parameters that indicate the architectural choices of a learner. For example, the maximum depth of a decision tree or the number of layers used in a neural network would be hyperparameters. However, some research uses "hyperparameter" as a term to indicate both the common understanding of hyperparameters, as well as including all possible ML system choices [BVR23]. For example, the choice of which data preprocessing methods to utilize is also considered a hyperparameter. Within this thesis, we will use "hyperparameters" to refer to the architectural choices of a learning algorithm, and we will refer to the possible ML system choices (aside from the hyperparameters of a learner) as the instantiation space.

Combining these two problems creates the Combined Algorithm Selection and Hyperparameter Optimization (CASH) problem. We then provide a formal definition of the CASH problem because it logically extends the Algorithm Selection Problem.

2.1.1 Algorithm Selection

This section presents the Algorithm Selection problem [Ric76]. To quickly summarize Rice's algorithm selection problem [Ric76]: it describes the algorithm selection problem as a "pipeline" of mappings from problem space to algorithm space. We first start with the problem space and use an extraction process to extract a feature space. The extraction process maps from the problem space to the algorithm space. Following this, we iteratively map performance until we find an algorithm that has the best performance. In essence, solving the algorithm selection problem is the first step toward automating the instantiation of an ML system. Figure 2.1. illustrates this process.

Definition 2.1. ***Algorithm Selection***

Let *Problems* be a problem space that contains all learning problems. Let $ex_{process}$ be an *extraction process* that extracts from *Problems* a feature space *Features*. Let *Algorithms* be an algorithm space that contains all candidate algorithms. Let *Performances* be the performance space representing the mapping from *Algorithms* to a performance $performance \in Performances$. We say that *Algorithm Selection* is the problem where for a given $problem \in Problems$, with features $ex_{process}(problem) \in Features$, we need to identify a mapping $S : Features \rightarrow Algorithms$ such that $S(ex_{process}(problem)) = algorithm \in Algorithms$ is the algorithm where the performance $performance(algorithm(problem)) \in Performances$ is maximized. □

2.1.2 CASH

Unfortunately, the Algorithm Selection Problem is incapable of expressing recent AutoML systems adequately. Without including a cost or budget, we cannot reduce AutoML to such a definition. Without a budget, any system could exhaustively search all possible ML system instantiations (assuming a finite number). Naturally, performing such a search is impossible in practice, and finding an optimal-performing instantiation is undecidable due

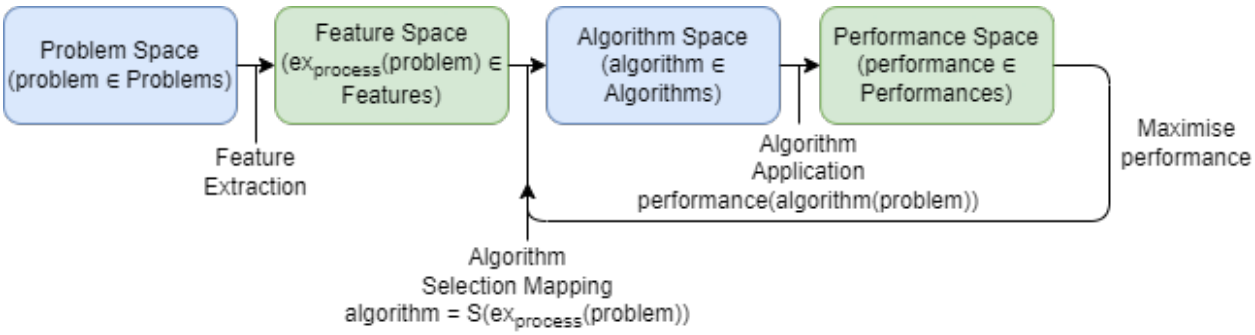

Figure 2.1: The pipeline of mappings required to select an algorithm which maximizes the performance for some problem.

to the infinite number of possible instantiations. Thus, including a mechanism to limit the breadth of the search is crucial. The budget of a system serves as this mechanism by specifying the allowed memory usage and/or time allotment during the search for appropriate ML system instantiations.

Furthermore, the Algorithm Selection Problem does not fully specify the hyperparameters for each algorithm. When instantiating an appropriate ML system, the chosen learner may have additional hyperparameters that specify its architecture. When creating a model, we can "tune" the learner's hyperparameters to make the model more accurate. Similarly, tuning the learner can decrease the time required for learning.

Thornton et al. [Tho+13] have defined AutoML in a way that is more expressive than the Algorithm Selection Problem. They describe AutoML as a *Combined Algorithm Selection and Hyperparameter* optimization problem. The CASH problem describes the process of taking a dataset, splitting it into two components (the training and testing sets), creating a model from the dataset, and then measuring the misclassification rate of the model. An AutoML system automatically performs these steps and attempts to minimize the misclassification rate of the created model.

Definition 2.2. ***CASH***

Let $\mathcal{A} = \{A_1, ..., A_{|\mathcal{A}|}\}$ be a set of algorithms.

For each $A \in \mathcal{A}$ there exists an associated hyperparameter space $\Lambda = \{\lambda_1, ..., \lambda_{|\Lambda|}\}$.

${A_\lambda}^{(i,j)}$ is an algorithm $A_i \in \mathcal{A}$ with an associated hyperparameter configuration $\lambda_j \in \Lambda$.

For some dataset $D = \{(((x_{1,1}, ..., x_{n,1}), y_1), ..., ((x_{1,m}, ..., x_{n,m}), y_m))\}$ we represent the training and testing sets produced from D as D_{train}, and D_{test}.

The misclassification rate of an algorithm with some hyperparameter configuration ${A_\lambda}^{(i,j)}$ on some dataset D is represented as $\mathcal{M}({A_\lambda}^{(i,j)}, D_{train}, D_{test})$

Given a dataset D, $\Phi(D) = \left\{\left({A_\lambda}^{(i,j)}, \mathcal{M}({A_\lambda}^{(i,j)}, D_{train}, D_{test})\right) \mid A_i \in \mathcal{A} \ \wedge \ \lambda_j \in \Lambda\right\}$ is the set containing all possible algorithms and associated hyperparameter configurations, and the misclassification rate of each when applied to D.

The combined algorithm selection and hyperparameter optimization problem is defined as finding $(a, b) \in \Phi(D)$ such that $\forall(a', b') \in \Phi(D), \ \ b \leq b'$. □

James et al. [Jam+13] summarize the reason for splitting the dataset into two components. They explain that the error rate of a given training algorithm should come from a dataset of previously unseen values (values not used to learn the model). To achieve this, an AutoML system must use two distinct datasets. If the system does not use two distinct datasets, it may over-fit [Roe+19]. An over-fit classifier might perform worse overall than a classifier that is not over-fit. The issue worsens because we cannot determine the rate of over-fitting. For example, in many applications, validation loss increases as training loss decreases past some threshold [GBC16]. As a result, an AutoML system might incorrectly infer that it is choosing better-suited ML system instantiations over its lifecycle, while it is actually making worse decisions.

AutoML systems use various methods of dataset interpretation, manipulation, and optimization to automatically and systematically create models for a given dataset. To achieve this, AutoML systems primarily search through a variety of learners and their hyperparameter configurations using various optimization methods. AutoML systems also automate the selection of "wrapper" components, such as feature preprocessing or data preprocessing, to lower the misclassification rate of a model.

Several of the terms within the Algorithm Selection problem and the CASH problem

are either not well defined or vague. We find that we need to formalize the concepts to understand the specifics of AutoML. For example, what is a dataset? How do we compute the misclassification rate of a classifier? In the following sections, we define various terms such as datasets, features, and classification.

2.2 Datasets

In this section, we focus on explaining datasets and provide a definition that we use to generate their representations. After defining datasets, we introduce definitions for the different categories of features that make up a dataset. We then present a definition of classification related to datasets, which we use as a basis for AutoML classification problems

2.2.1 Dataset Definition

A common approach in both research and industry is to use a 2-dimensional matrix to represent feature-values, similar to relational database models [Cod70]. In this matrix, we define a *Dataset* as a table of values. We use columns to represent the feature-values of the dataset coming from a *Feature* and rows to represent the *Examples*.

We define a set of features, referred to as *Feat*. Each feature within *Feat* is a set containing the possible feature-values for that feature. We then state that a dataset comes from a specific *Feat* set. The feature-values present within a given dataset are contained within their corresponding feature. However, not all feature-values contained within a feature need to be present within the dataset.

For example, a dataset describing a set of vehicles can be composed of a *Feat* set containing the features "Is_Automatic" and "Mileage." In this instance, the feature "Is_Automatic" contains {"True," "False"} as possible feature-values, while the feature "Mileage" contains all real numbers. By using this *Feat* set, multiple datasets can be represented. For instance, the dataset (("True", 12.8), ("False", 7)) can be represented using the provided *Feat*.

In this case, the first example in the dataset has the feature-values "True" for the feature "Is_Automatic" and 12.8 for the feature "Mileage," representing an automatic vehicle with a mileage of 12.8.

A nuance of describing a dataset is that we can use the *Feat* set to produce multiple datasets. In the previous example, there were two examples within the dataset. However, we can describe additional datasets in the same way. For instance, the dataset (("True", 12.8),("False", 7),("False", 15.5)) serves as another example. With this example we see that the previous dataset is a subset of the current dataset. We can also represent a dataset that does not intersect the previous examples at all. For example, the dataset (("False", 12.8),("True", 7)) does not intersect with the two previous datasets. Although some examples in the dataset share common feature-values with the previous datasets, none of them are identical. Therefore, this dataset does not intersect with the previous datasets.

Definition 2.3. ***Dataset***

Let $Feat = \{F_1, ..., F_n\}$ be a feature set where $n \in \mathbb{Z}^+ \cup \{\infty\}$ denotes the number of features. Each feature F_i for $i \in 1, ..., n$ is defined as $F_i = \{fv_{i,1}, ..., fv_{i,k}\}$ where $k \in \mathbb{Z}^+ \cup \{\infty\}$ and $fv_{i,j}$ represents a specific feature-value belonging to F_i.

Let $F_{label} = \{l_1, ..., l_j\}$ where $j \in \mathbb{Z}^+ \cup \{\infty\}$ and l_i represents a specific label.

The set $D = \{((x_{1,1}, ..., x_{n,1}), y_1), \ldots, ((x_{1,m}, ..., x_{n,m}), y_m)\}$ is a dataset over *Feat* where $\forall x_{a,b}\, \exists fv_{a,c} \in F_a$ where $x_{a,b} = fv_{a,c}$ and $\forall y_d\, \exists l_e \in F_{label}$ where $y_d = l_e$. Note that m denotes the number of examples found within D.

Each $((x_{1,i}, ..., x_{n,i}), y_i)$ in A is an example in A.

D_i is used to isolate a specific example from A such that $D_i = ((x_{1,i}, ..., x_{n,i}), y_i)$.

□

In this context, n and k can be infinite. This means the feature set *Feat* may contain infinitely many features, and each feature may have infinitely many possible values. Such a setup provides a broad and flexible representation of features and their feature-values,

accommodating a wide range of scenarios, including theoretical models, continuous features, or cases involving unbounded data.

A feature is an abstract concept that defines which feature-values can appear within a dataset, thereby constraining it. Datasets created from a specific *Feat* set are limited by the features within *Feat*. The interpretation of a dataset depends on the *Feat* set it uses, as the same dataset can be understood differently depending on the *Feat* set. For example, a *Feat* set with a feature representing mileage in litres per kilometre will lead to a different understanding of the dataset compared to a *Feat* set where mileage is measured in litres per mile.

We highlight the constraints on a dataset because, within the field of AutoML, an AutoML system will infer the *Feat* set by observing the feature-values in a dataset. Knowledge pertaining to the *Feat* set a dataset is composed from is not given in common scenarios. Take, for example, the AutoML ChaLearn competitions [Guy+19]. Within these competitions, systems are only given limited knowledge about a dataset apart from the specific feature-values. During a competition, a system is only told whether a feature is numerical or categorical. In Section 2.2.2, we discuss features and what it means to be categorical or numerical.

A label is a value or category assigned to each example found within A. The labels for a dataset are contained within their own feature denoted as F_{label}. Each example in a given dataset has an associated label which is used to categorize the example. For example, in our vehicle dataset example, a label for each example could be "True", or "False" if F_{label} is the "Is_Automatic" feature.

F_{label} is not considered to be a member of *Feat* as it is conceptually different from a feature. Rather, observations about the feature-values of a dataset are used to infer the label associated with each example. In Section 2.2.3. we discuss the importance of labels.

Our formal definition of a dataset can be visualized as a two-dimensional array. In this array, each row represents an example, while each column corresponds to feature-values

from a specific feature. The set of features acts as column headers (also known as attributes [Aha92]). Each example (also known as a row, entry, sample, or instance [Aha92]) contains a vector of feature-values, with each value belonging to a particular feature.

$$\begin{matrix} F_1 & \dots & F_n & F_{label} \end{matrix}$$
$$\begin{bmatrix} ((x_{1,1} & \dots & x_{n,1}), & y_1) \\ ((x_{1,2} & \dots & x_{n,2}), & y_2) \\ \vdots & \ddots & \vdots & \vdots \\ ((x_{1,m} & \dots & x_{n,m}), & y_m) \end{bmatrix}$$

It is important to note that although we define datasets this way this is not the only way to represent them. Graph-like datasets [JV13] and time-series datasets [EA12], among others, are also valid representations of datasets. However, AutoML predominantly uses the 2D, tabular structure for datasets we have described above. Because of this, we will only consider this representation.

2.2.2 Features

It is beneficial to define what a feature is for this thesis because features are the primary way knowledge is extracted from a dataset. The prior knowledge that comes with knowing the set of feature-values present in a given F_i acts as a constraint on a dataset by constraining which possible feature-values can be present for a given example. Conversely, if a specific feature F_i is not known then a system is only able to infer knowledge based on observation and/or extrapolation.

Individual feature-values alone are often insufficient for drawing meaningful conclusions. However, when we consider all feature-values of a feature together, we gain a better understanding of the dataset. Based on the different types of features, there are four levels of measurement that indicate how precisely feature-values are recorded: *nominal*, *ordinal*, *interval*, and *ratio* values [Kir08].

Nominal features are those where feature-values can only categorized into a finite set of possibilities. These feature-values can be grouped but not inherently ordered or measured in terms of arithmetic operations such as addition or subtraction. Operations on nominal feature-values require defining the entire set of values to make any operation meaningful. If a feature is defined, it is possible to create custom operations by mapping feature-values to other desired sets.

Binary features are a subset of nominal features with exactly two possible values. With only two values, these features can be represented using Boolean symbols. This distinction allows systems to make informed decisions based on the binary nature of the feature. For instance, in the vehicle dataset, the feature "Is_Automatic" was a Boolean value. If a feature-value deviated from true or false, it would indicate an anomaly in the dataset.

Ordinal features are more precise than nominal features as they can be categorized and ranked (i.e., ordered). However, the exact distance between ordered feature-values is not defined. For example, the Likert scale [Lik32] used in psychology ranks responses (e.g., very dissatisfied to very satisfied) without quantifying the distance between them.

Interval features are similar to ordinal features but with meaningful and consistent differences between feature-values. While continuous features theoretically offer an infinite number of values between any two points, the intervals between feature-values remain equal in the context of interval scales. The concept of equal intervals refers to the measurement system's consistency rather than practical precision. Interval features allow for algebraic computations, such as calculating information gain [Ken83] or deriving statistical metrics from a dataset.

Ratio features are a subset of interval features that include a true zero point, indicating the absence of the property being measured. For example, while calendar dates are interval-scale features with equal intervals, they lack a true zero point and thus cannot indicate relative magnitudes (e.g., "twice as early"). In contrast, measurements such as weight or length have true zero points, allowing for meaningful ratios between feature-values.

A dataset may also include feature-values that do not fit into these four levels of measurement. These values typically represent missing data, which might occur due to unobserved or uncollected data (e.g., equipment failure). In this thesis, we denote each such exception as $\perp$. If $\perp$ is a possible feature-value, the associated feature in the *Feat* set must also include $\perp$.

Previously, we noted that AutoML systems are designed to operate without complete knowledge of the feature set used to compose a dataset. Information about the level of measurement for a feature is often categorized as either "numerical" or "categorical." In terms of measurement levels, this classification roughly corresponds to nominal-level or ratio-level features. For this thesis, we will adopt the standard approach of categorizing features as either categorical or numerical [Guy+19]. However, a system can also infer if a categorical feature is also binary by counting the number of possible feature-values present in a column. If there are only two distinct feature-values, an AutoML system will infer that the corresponding feature is binary.

Definition 2.4. ***Feature-Properties***

Let $\perp$ be a symbol that represents an unknown value.

A feature $F_i = \{fv_{i,1}, ..., fv_{i,k}\}$ is finite if $k < \infty$, and infinite otherwise.

A feature is numerical if it is infinite and if $\forall fv \in F_i$, $fv \in \mathbb{R} \vee fv \in \{\perp\}$.

If a feature is finite, or $\exists fv \in F_i$, $fv \notin \mathbb{R} \wedge fv \notin \{\perp\}$ then the feature is categorical.

If a feature is categorical and $k = 2$ we further specify that the feature is binary.

A feature is considered to contain an unknown value if there exists a $fv \in F_i$ where $fv = \perp$. □

2.2.3 Dataset Classification

Researchers use the concept of a learning task to define the purpose of a given algorithm [MRT18]. Standard tasks include classification, regression, ranking, clustering, and manifold learning [MRT18]. Within AutoML, researches predominantly use classification and regression tasks [Guy+19]. However, for our proof of concept, we chose to specifically focus on

classification to maintain a reasonable scope of work.

The task of classification involves assigning a category or "class" (label) to an example from a dataset. Although our definition includes the labels of each example, a key component of classification is the ability to assign a label to an example that does not already have an associated label. In other words, a classifier predicts the label for an example. We term an example with an associated label "known," and we term one without an associated label "unknown."

To perform classification, a learning algorithm (learner) examines a dataset that contains associated labels. The learner then uses these observations to generate a structured knowledge representation. This representation infers characteristics about an example and predicts its label. The learned representation for any learning task is known as a model. When performing classification specifically, a model is referred to as a classifier.

Mohri et al. [MRT18] further specify that ML is fundamentally concerned with generalization. For instance, in a typical classifier learning scenario, we use a finite set of labeled examples to make accurate predictions about new, unseen examples. Learning is the process of choosing a function from what we call a hypothesis set. We define a hypothesis set as a subset of all possible functions, particularly those that map from examples to labels. We then employ the chosen function to label all instances, including those that we did not include in the original training set.

Definition 2.5. ***Classifier***

Let D be a dataset composed over *Feat*, and let F_{label} be the labels which can occur in D.

Let *Examples* be the set of all possible examples which can be composed from *Feat*.

A classifier is a function with the following signature $Classifier : Examples \rightarrow F_{label}$ □

A learner is an algorithm that uses a dataset to generate a model of that dataset. This is the core of most ML systems as a learner dictates how it interprets and represents knowledge within a model. The job of a learner is to observe a dataset and to generate structured knowledge that predicts labels for the examples in that dataset. In the context of a hypoth-

esis set, a learner selectively determines which function within the hypothesis set is most appropriate for the given task.

In practice, selecting a specific classifier function is complicated. We use inferred knowledge, gained by analyzing patterns and relationships within the dataset, to build structured knowledge representations such as decision trees, neural networks, or probabilistic models. These representations encapsulate the relationships and dependencies that the learner identifies during the learning process. By organizing information in a structured way, learners can efficiently navigate the hypothesis space to select the most appropriate classification function.

Definition 2.6. ***Learning Algorithm***

Let $\mathcal{D}$ be the set of all datasets that can be composed over *Feat*, and let F_{label} be the labels which can occur in any $D \in \mathcal{D}$.

$\mathcal{H} = \{Classifier | Classifier : Examples \rightarrow F_{label}\}$ is the hypothesis set.

A learner is a function with the following signature $\mathcal{L} : \mathcal{D} \rightarrow \mathcal{H}$. □

Once a learner has created a classifier we can query the classifier to predict labels for examples. A learner generalizes to unknown examples by selecting a specific classification function from within the hypothesis set. In practice, however, if an unknown example lacks a label, how can we be certain that the classifier accurately predicts the label? Although we cannot achieve absolute certainty regarding a classifier's predictive capabilities, we can still measure the classifier's accuracy.

After splitting a dataset into two, a training set and a testing set, a learner is applied to the training set, and the produced classifier predict labels for the testing set. The accuracy of a classifier is the percentage of examples for which the classifier predicts labels accurately.

Definition 2.7. ***Accuracy***

Let D be a dataset composed over *Feat*. Let F_{label} be the labels which can occur in D. Let D_{train} contain some of the examples in D, and let D_{test} contain the others such that D_{train} and D_{test} are disjoint.

Let $Classifier$ be some classifier produced by applying some learner L to D_{train} such that $C = L(D_{train})$

$Accuracy = \frac{1}{m} \sum_{n=1}^{m} \mathbb{I}(Classifier(D_{test_n}), y_n)$ where $\mathbb{I}$ returns 1 when both the classifier's output and the true label (y_n) are the same, and 0 otherwise. □

The accuracy metric is a straightforward way to quantify a classifier's performance. The accuracy metric closely relates to the misclassification rate, which complements performance measurement. The misclassification rate is defined as the proportion of examples in the testing set that are incorrectly classified. Since every example is either classified correctly or incorrectly, the sum of the accuracy and misclassification rate must equal 1.

An ML system is a structure that determines how a learner should be utilized. In practical terms, a system can perform additional tasks that a learner does not, helping to produce a better-performing model. A system may use methods such as dataset transformations and encoding to give a learner a more suitable representation for creating a model. Similarly, an ML system may also decode a prediction into a specific label. Ultimately an ML system acts as a "wrapper" for a learner and may also add extra structure around the produced model.

2.3 Ontologies

In this section, we focus on discussing ontologies as tools for sharing and structuring knowledge across diverse domains, as Stumme emphasizes [Stu02]. Stumme specifically mentions using 'business knowledge' and focuses on incorporating additional knowledge from within a specific business domain. Stumme means that knowledge represented in an ontology spans a spectrum of generality.

Afsharchi et al. [AFD06] note that, within computer science, "ontology" is a vague and overused term without a universally agreed-upon definition. The multitude of ontologies, each with distinct definitions and applications such as KL-One [WS92], LOOM [Mac91], and FLogic [KLW95], reflects this ambiguity. The sheer volume of ontologies has led to developing

ontological languages like Web Ontology Language (OWL) [AH09] and Ontology Inference Layer [Fen+00], which facilitate conversion between different ontological frameworks.

To avoid potential confusion, we provide a concise definition, originally proposed by Stumme, that serves as a core ontology [Stu02]. This foundational understanding lays the groundwork for applying ontologies in our discussion of Ontology-Enhanced AutoML in the next chapter.

2.3.1 Ontology Definition

A core ontology consists of a structured framework with a set of concepts and a set of relations, which are separate from the concepts. It includes a concept hierarchy, a partial ordering of concepts, and a relation hierarchy, a partial ordering of relations. We associate each relation with a signature, a set of concepts, and order relations based on the size and ordering of their signatures. Specifically, we consider a relation less than or equal to another if both have signatures of the same size and each concept in the first relation's signature is less than or equal to the corresponding concept in the second relation's signature. Additionally, if one concept is less than or equal to another in the concept hierarchy, we consider it a sub-concept of the other.

Definition 2.8. ***Core Ontology***

Let $Concepts$ be a set of concept identifiers (referred to for brevity simply as concepts), and $Relations$ be a set of relation identifiers (referred to as relations) which is disjoint from $Concepts$.

A core ontology is a structure $\mathcal{O} := (Concepts, \leq_{\mathcal{C}}, Relations, \leq_{\mathcal{R}}, \sigma)$ where σ is a function $\sigma : Relations \to Concepts^{+}$ which maps from relations to one or more concepts, called a signature.

$\leq_{\mathcal{C}}$ is a partial order on $Concepts$ which we call a concept hierarchy.

$\leq_{\mathcal{R}}$ is a partial order on $Relations$ called a relation hierarchy where the order $r_1 \leq_{\mathcal{R}} r_2$, $r_1, r_2 \in Relations$ implies that $|\sigma(r_1)| = |\sigma(r_2)|$ and that for any projection $\pi_i (1 \leq i \leq$

$|\sigma(r_1)|$) of the resulting vectors $\sigma(r_2)$, or $\sigma(r_1)$, $\pi_i(\sigma(r_1)) \leq_{\mathcal{C}} \pi_i(\sigma(r_2))$ holds.

If $c_1 \leq_{\mathcal{C}} c_2$ where $c_1, c_2 \in Concepts$, then c_1 is called a *sub-concept* of c_2. □

In this context, projections mean checking whether each concept from one relation's signature matches with a concept from another relation's signature while respecting the generalization condition. If all the conditions are satisfied, the relation hierarchy is valid.

The relation hierarchy is used to specify the relationships between concepts within an ontology, and their ordering. Figure 2.2. shows how ontologies are typically depicted. The provided figure is relatively simple; more complex ontologies are depicted with more complex graphs.

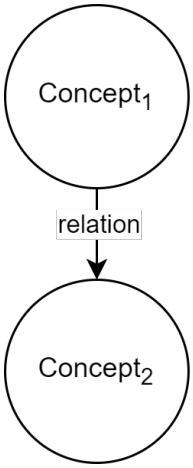

Figure 2.2: How ontologies are visually represented. Circles indicate a concept while arrows indicate relationships between concepts.

To illustrate this definition we provide an example.
Let $Concepts = \{Math, Algebra, Geometry, History, Sociology\}$, and let $Relations = \{prerequisite, related\_to\}$. The concept hierarchy is defined as $Algebra \leq_{\mathcal{C}} Math$, and $Geometry \leq_{\mathcal{C}} Math$. The signature function $\sigma(prequisite) = \{Algebra, Geometry\}$, and $\sigma(related\_to) = \{History, Sociology\}$. If we define *prequisite* to be more specific than *related_to* (because prerequisites are a more detailed kind of relationship) then we can say $prerequisite \leq_{\mathcal{C}} related\_to$.

2.3.2 Symbolic Concepts

Afsharchi et al. further specify the use of ontologies in ML called symbolic concepts. Symbolic concepts represent a set of objects that share feature instantiations.

Definition 2.9. ***Symbolic Concepts***

Let $\mathcal{U}$ be the universal set of all features.

The set of symbolic concepts $Concepts_{sym} = \{Feature | \exists F \in \mathcal{U}, Feature \subseteq F\}$. □

The definition of symbolic concepts describes them as a form of a feature set.Within the symbolic concepts feature set, however, we may impose additional constraints on which feature-values are allowed. For example, a symbolic concept describing animals may constrain a given feature to include only those animals that have ever occurred in nature. By defining symbolic concepts this way, we enable matching symbolic concepts to a dataset by comparing the feature sets that compose each.

Definition 2.10. ***Dataset to Symbolic Concept Matching***

Let $Feat$ be a feature set, and let D be a dataset composed from $Feat$.

D matches a symbolic concept $C_{sym} \in Concepts_{sym}$ if and only if $\forall F \in C_{sym}, F \in Feat$. □

Our definition of matching checks whether the dataset contains the same conceptual features and whether the constraints imposed on the symbolic concept also appear in the dataset.The dataset must include all of the features contained within a symbolic concept. This approach creates a direct mapping from the dataset to the symbolic concept. If the dataset lacks some features from a symbolic concept, we cannot map the dataset to that symbolic concept.The symbolic concept is defined by its features; thus, if the dataset lacks some of these features, we cannot determine whether the symbolic concept pertains to the dataset. If the dataset contains only a subset of a symbolic concept's features, the symbolic concept essentially refers to a different concept not found in the dataset.

Dataset-to-symbolic-concept matching allows us to connect an ontology to a dataset.We can match any symbolic concept that includes all its features in the dataset's feature set to the dataset. This process allows us to utilize the knowledge from the ontology within the dataset.We can incorporate the knowledge from the ontology into the dataset for use in ML. Thus, symbolic concepts bridge the dataset and the ontology.

2.3.3 Concepts

However, we encounter an issue within the structure of ontology relationships when deriving knowledge from an ontology for a dataset. Specifically, we face the issue of how to procedurally utilize the knowledge in an ontology in a decidable manner. Stumme's general definition of an ontology includes ontologies that we consider undecidable [Bor96][Ros06]. Although researchers have done much work to develop ontological languages that are decidable [Mot05], their attempts to produce both decidable and useful ontological languages have not succeeded.

We can address the undecidability issue of ontological logic representations by using additional knowledge. When implementing something specific with an ontology, we introduce specific mechanisms to represent particular knowledge. For example, we can incorporate functions and predicates (functions that produce a Boolean value) into an ontology to make querying procedural.

To represent the additional knowledge associated with a concept, we define a generalized concept as follows. In this framework, we define a concept as
$\mathcal{C}k = (Csym, Additional_k)$ where $C_{sym} \in Concepts_{sym}$, and $Additional_k$ represents the set of additional knowledge associated with $\mathcal{C}_k$. In this way, we define an ontology within this thesis as an extension of Stumme's ontology that includes additional information.

2.4 Computational Geometry

Within our experiments, we specifically use geometric objects to enhance AutoML with ontological knowledge. The representation of knowledge within geometric objects varies greatly depending on various factors. Reviewing computational geometry is daunting because many methods exist for modeling topological geometric data.

We ground our experiments fundamentally in terms of physical laws. Instead of discussing the variety of approaches, we will not consider constructs that exist in higher dimensions or are mathematically 'interesting' for our purposes. Instead, we will focus on representations commonly used in many fields where people construct objects for material purposes. This approach lets us consider only a very limited set of representations. We use three representations: the boundary representation, the implicit representation, and the point cloud representation. Figure 2.3 shows a visual representation of the three approaches.

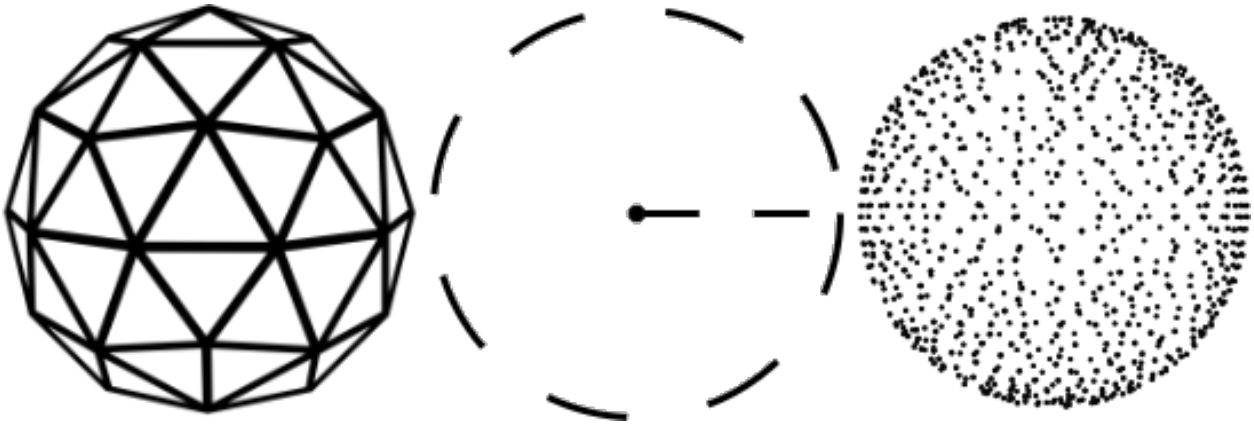

Figure 2.3: Three computational geometry representations. On the left is boundary representation where an object is defined by a series of vertices connected by edges. In the middle is implicit representation which is defined as a single point with an implicitly defined shape of a sphere dictated by a radius. On the right is a point cloud that is represented by points on the boundary of an object.

2.4.1 Boundary Representation

Perhaps the most common approach for geometric modeling is the boundary representation (b-reps) [Str06]. By using primitives such as vertices and edges, we can define a geometric model in terms of its surface boundary. We use vertices to determine points on the surface of an object, and edges encode the connections between vertices.

Within this encoding scheme, many variations exist. Winged-Edge [Bau75], Face-Vertex [ADF85], and Half-Edge [Bot+02] encodings are just a few examples, with Face-Vertex being the most commonly used in popular file formats such as FBX, STL, OBJ [MB08], and DWG [PY16]. These encodings differ in how they determine the planar surface of faces as expressed by the vertices and/or edges. Ultimately, these encodings work to create a topological surface of an object using polyhedra as bounds.

The primary advantage of a boundary representation is its graph-like structure. Researchers adapt and implement efficient algorithms from graph theory into computational geometry for objects represented using boundary representations [Pac99]. Creators can quickly modify models with simple operations pre-programmed into 3D modeling software.

2.4.2 Implicit Representation

Implicit representation encodes volumes as equations [Gom+09]. We often use this type of encoding for rendering via ray-tracing [Gla89] because calculating intersections is straightforward. Additionally, since the equations are continuous, we can achieve arbitrarily high resolution without extra storage overhead. However, modifying the underlying geometry requires significant work, as it involves defining complex equations for intricate topology. Although less common than b-reps, the implicit representation encoding scheme offers many benefits [Mue20]. By combining primitive shapes through functions, modelers can create complex geometry more efficiently. Many modern animation films use implicit representation because of the power of graphical processing units in computers [Wal+09].

2.4.3 Point Cloud Representation

One of the more naïve methods for modelling geometry is the point cloud encoding. As the name suggests, point cloud encoding involves a collection of three-dimensional points [RC11]. We see this method becoming more prevalent as three-dimensional scanning applications become more commonplace. Since this encoding lacks any concept of ”inside” or ”outside” of an object, it serves as a rough approximation of b-reps encoding. Thus, we typically transform point clouds to convert them into direct b-reps representations [Rem03].

2.5 ML and its Variants

In this section, we focus on distinguishing various ML variants such as AutoML, PSML, AutoDL, and Knowledge-Enhanced ML. Since we have already introduced AutoML, we will first discuss different AutoML frameworks to help the reader understand how an ML system, a learner, and a classifier work in practice. Next, we will compare AutoDL, Knowledge-Enhanced ML, and PSML to examine the different approaches to ML.

AutoML Framework

AutoML frameworks commonly describe a pipeline of individual components. More advanced AutoML systems use a greater number of well-known components. However, many earlier systems in previous research use only one or two components. We generalize from multiple overview comparisons to describe a singular, generalized pipeline [NP19][Guy+19][BVR23][NP19]. Our generalized pipeline includes Metalearning, Data Preprocessing, Feature Preprocessing, Classifier Selection, Optimization, and Postprocessing components. Figure 2.4 depicts the underlying components of an AutoML framework.

Metalearning in AutoML systems involves incorporating previously learned knowledge into a current learning task. Vanschoren [Van19] summarizes metalearning as “systematically observing how different machine learning approaches perform on a wide range of learning

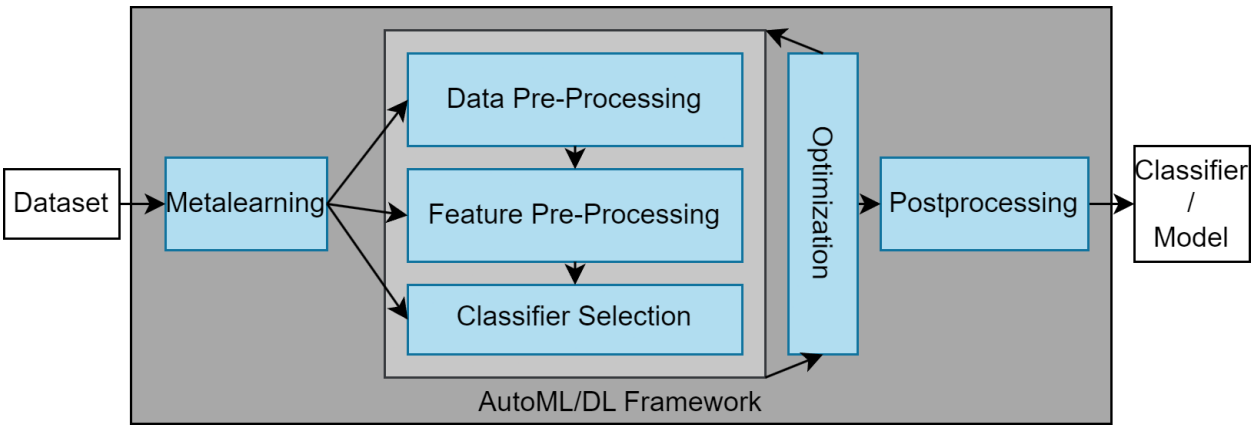

Figure 2.4: The AutoML framework.

tasks, and then learning from this experience." In the sequence of events performed during AutoML, we use metalearning as the first step. This is because metalearning informs an AutoML system about which hyperparameters of the other components are likely to yield better results. Thus, this component acts as a "beacon" that helps guide an AutoML system to instantiate better-performing ML systems.

Systems accomplish metalearning by referencing a set of pre-learned datasets when automating the creation of a classifier for a given dataset. By computing the metafeatures of a dataset, these systems can reference previously learned knowledge to gain a better understanding of how to implement a search over the different ML system instantiations available to them.

Metalearning can also utilize a technique called warmstarting. Warmstarting accelerates the optimization process by initializing a learner with specific parameters or knowledge structures from a previously trained classifier. Instead of starting from scratch, warmstarting leverages knowledge gained from prior ML system instantiations, potentially reducing the computational resources and time required for optimizing multiple ML system instantiations. Warmstarting can take various forms depending on the specific AutoML framework. One common approach involves using a learner with hyperparameters similar to those used by a previously instantiated ML system that performed well on a similar dataset.

In essence, metalearning allows an AutoML system to quickly find suitable ML system instantiations likely to work well for learning a model [Feu+15]. By utilizing previously learned knowledge, an AutoML system can make better-informed decisions about which ML system instantiations are likely to perform well for a given dataset. Metalearning informs many future components in a pipeline about potential starting points from which to begin the search.

After metalearning, data preprocessing is the next step in the generalized AutoML framework. Since an AutoML system should function even with arbitrary datasets, the specific feature-values within an input dataset can vary. A dataset might also contain discrepancies that could lead to erroneous learning. Additionally, the feature-values of a dataset might not be encoded in a manner suitable for certain learners used in the Classifier Selection component discussed later. Thus, data preprocessing involves techniques such as normalization, feature standardization, or missing-value patching. These techniques transform a dataset into a more easily understood format. Essentially, data preprocessing directly modifies the input dataset to make learning easier or more robust. A list of data preprocessing methods appears in Table 2.1.

The Feature Preprocessing component follows directly after the Data Preprocessing component. If Feature Preprocessing occurred earlier, the manipulation of feature-values done in Data Preprocessing would be wasted. Feature Preprocessing works to extract new knowledge about a dataset from its features. For example, dimensionality reduction is a common feature preprocessing technique found in many AutoML systems. Algorithms like Principal Component Analysis (PCA) [WEG87] and Linear Discriminant Analysis (LDA) [XPT13] help "distill" the knowledge represented by a dataset. Table 2.2. outlines the various feature preprocessing methods used across the described systems.

In contrast to data preprocessing, which directly modifies the feature-values of a dataset, feature preprocessing keeps the feature-values of the dataset unaltered. Instead, feature preprocessing may append new features to the dataset. Alternatively, in cases such as

Input Rescaling	Rescaling numerical values to either have a mean value of 0, to be mapped to the range [0,1], or to be normalized to have identical magnitudes (in the event of vector values).
Encoding	Encoding allows for replacing a set of categorical features $F = \{v_1, v_2, ..., v_k\}$ to a set of numerical values. For example, a common approach (one-hot encoding) transforms the set of features to the set of the powers of 2. i.e. $E(F) = \{2^1, 2^2, ..., 2^k\}$.
Imputation	Replacing missing values with others such as with the mean, or mode of a set of numerical feature values.
Balancing	When certain algorithms allow for the weighting of classes, these weights are set based on the class imbalances of the predictor feature.
Data Augmentation	Creating more training examples by slightly modifying existing ones. For example, the rotation or inversion of images when using deep learning.

Table 2.1: A list of common data preprocessing methods used among various AutoML systems [NP19][Guy+19][BVR23][KKK22].

Matrix Decomposition	A method used to decompose the dataset into maximally distinctive features. PCA, LDA, linear SVM, and fast ICA are such examples.
Univariate Feature Selection	Selecting for a subset of all features based on some statistical metric. An example of this would be selecting the 10th percentile of classes with regard to feature variance.
Classification-based Feature Selection	Selecting for a subset of all features based on some statistical metric. An example of this would be selecting the 10th percentile of classes with regard to feature variance.
Feature Clustering	Merging features to produce new features.
Kernel Approximation	Rather than apply some kernel (a function that implicitly maps data into a higher-dimensional space) onto a dataset, kernel approximations are used instead. These approximations can be applied to transform the input dataset or to increase the dimensionality while not requiring as much computational time.
Polynomial Feature Expansion	Expands the provided feature set by calculating all n-degree polynomial combinations of the features. Here n is a fixed, preset value.
Feature Embedding	The mapping of each example in the input dataset to some lower dimensional embedding. This was only performed by Feurer et al. [Feu+15] and was done via random tree embedding.
Sparse Representation Transformation	Some transformation function that maps a dataset from an input space of relatively low density to one of relatively higher density.

Table 2.2: A list of common Feature Preprocessing methods used among various AutoML systems [NP19][Guy+19][BVR23][KKK22].

dimensionality reduction, the system discards the original dataset in favor of a new dataset containing any newly generated features.

Feature preprocessing is crucial for the Optimization component to work effectively in AutoML systems. As Nagarajah and Poravi [NP19] noted, feature preprocessing significantly influences the success of learning. Consequently, developers often fine-tune this component more carefully when creating AutoML systems.

Following the Feature Preprocessing component, the Classifier Selection component takes over. Classifier selection is the most complex task performed by AutoML systems, as different classification tasks require different types of classifiers. Some classifiers cannot handle certain types of data. For example, decision trees often perform worse with strictly numerical feature-values, while support vector machines perform better with numerical feature-values. Although transforming a dataset during preprocessing can mitigate this issue, there is still a risk of losing knowledge during transformation. For instance, converting a feature into a nominal one might enable the use of a decision tree learner, but it may also lose valuable information. Thus, it is crucial for an AutoML system to correctly identify which algorithms are more suitable for a given task. By selecting an appropriate learner, an AutoML system can leverage the dataset's inherent characteristics, potentially leading to improved model accuracy.

Different classifiers also vary in terms of computational requirements during training. In AutoML, where efficiency is critical, selecting a classifier that balances accuracy and computational resources is vital. For instance, simpler learners might train faster but produce classifiers with lower accuracy. Conversely, more complex learners may require significantly more computational resources but create classifiers with higher accuracy. This is why data preprocessing and feature preprocessing must occur before classifier selection. Without the additional knowledge from these preprocessing steps, identifying and selecting a suitable learner becomes more challenging.

After selecting a classifier, the system utilizes the Optimization component. The Opti-

mization component's job is to select ML system instantiations likely to produce classifiers that outperform models created by previous instantiations. From a general perspective, the Optimization component is responsible for selecting ML system instantiations at the start of each iteration. Since an ML system determines how a learner should be utilized, this perspective is fitting. To optimize, a system chooses ML system instantiations and trains them to create classifiers iteratively over the AutoML system's runtime.

The Optimization component decides which preprocessing methods to use and selects the learner for creating the next classifier. For example, the Optimization component might choose input rescaling and imputation as data preprocessing methods, matrix decomposition as a feature preprocessing method, and C4.5 with a max tree depth of 15 as the learner for a given dataset. Ideally, these choices are well-informed, based on knowledge gained from warmstarting, rather than made randomly.

Optimization in an AutoML system often involves balancing exploration and exploitation of the hyperparameter and instantiation space. Advanced search optimization algorithms, such as Sequential Model-Based Optimization for General Algorithm Configuration (SMAC) [HHL11] and Bayesian Optimization with Hyperband (BOHB) [FKH18], may efficiently search the hyperparameter and instantiation space to discover optimal configurations. Techniques like Bayesian optimization with Tree-of-Parzen Estimators (TPE) [Ber+11] or successive halving may dynamically allocate computational resources to promising configurations, improving the optimization process's efficiency.

Many AutoML systems allow users to choose the search optimization algorithm during classifier creation. Given the variable nature of AutoML, some users may prefer exploration while others prefer exploitation. The use cases of the final ensemble classifier depend entirely on the user's needs. Therefore, allowing users to specify their preferences is advantageous. Table 2.3. lists the specific search optimization algorithms commonly used across many systems.

Auto-Sklearn [Feu+15] is one of the most widely used AutoML systems among the general

SMAC	Sequential Model-Based Optimization for General Algorithm Configuration. A Bayesian Optimization method that works well when sampling is a computationally expensive operation.
Successive Halving	Successive halving keeps the best half of the hyperparameter configurations and discards the rest. It will continue until a singular hyperparameter configuration is left. Successive halving requires that hyperparameter configurations can be terminated prematurely and evaluated at indeterminate times.
Hyperband [Li+18]	A form of Successive Halving that dynamically allocates time for each hyperparameter configuration based on the previous iteration's results.
BOHB	Bayesian Optimization with Hyperband. Dynamically allocates training time for each hyperparameter configuration based on estimations computed by Bayesian Optimization sampling.
Grid Search	Iterative searching over states in grid-like steps. Similar to path-finding algorithms with the exception that each state is a hyperparameter configuration as opposed to coordinates.
Random Search	Randomly searching over hyperparameter configurations.
TPE	Tree-of-Parzen Estimator. A Bayesian Optimization method that uses both a probability distribution of good and bad estimations of hyperparameter configurations.
GA	A Genetic Algorithm that iterates over hyperparameter configuration trees of all components for a tree of genetic primitives.

Table 2.3: A list of common search optimization algorithms used among various AutoML systems [NP19][Guy+19][BVR23][KKK22].

public. Auto-Sklearn has won every AutoML ChaLearn competition to date, demonstrating the effectiveness of AutoML for learning from arbitrary datasets [Guy+19]. Feurer et al. were the first to incorporate two additional components into Auto-Sklearn: metalearning for warmstarting a Bayesian optimizer [Bra+08][LH18], and ensemble creation [Car+04][Pol12] of trained configurations. Figure 2.5. visually depicts the Auto-Sklearn system.

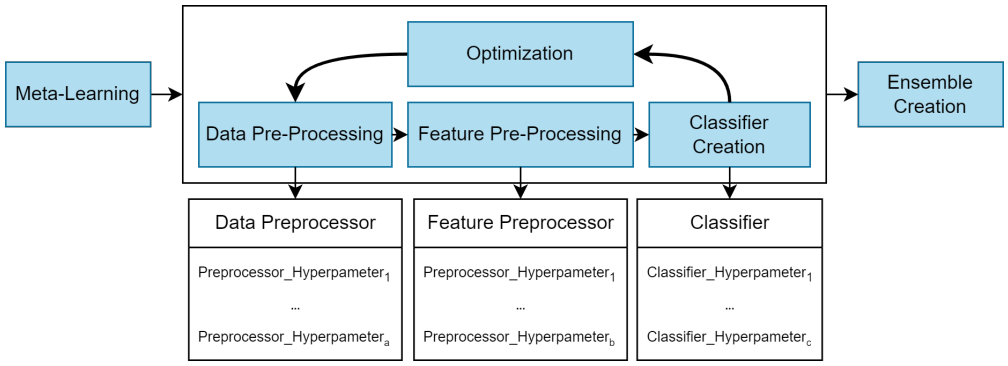

Figure 2.5: The components of Auto-Sklearn [Feu+15]. Auto-Sklearn creates a sequence of steps that acts as an AutoML pipeline. Notice how the typical Optimization iteration is contained within, and that Ensemble Creation and Metalearning are considered external steps.

AutoML systems that existed before Auto-Sklearn used search methods such as random search or grid search to explore the hyperparameter and instantiation space. These approaches are computationally intensive, as they require iterating through numerous combinations that may or may not be productive. Feurer et al. improved this process by creating an internal repository of 140 pre-learned datasets. These datasets had their metafeatures recorded alongside the hyperparameter and instantiation parameter values that produced the best-performing classifiers. When a new dataset is provided to Auto-Sklearn, the system computes the metafeatures of the dataset and compares them to the metafeatures of the pre-learned datasets using the L1 distance [Bla98]. By selecting a learner, preprocessing methods, and associated hyperparameter configurations based on the pre-learned datasets, Auto-Sklearn begins with a better-informed starting point during optimization.

Feurer et al.'s incorporation of ensemble creation is also key to the superior accuracy of Auto-Sklearn compared to its competitors. During optimization, the system keeps a

record of all classifiers created in each iteration. These classifiers are aggregated into an ensemble at the end of the search, forming a final ensemble classifier. Feurer et al. used the ensemble selection method [Car+04] for ensemble creation, which outperformed stacking [Wol92] during their development tests.

We have generalized the existing AutoML framework slightly, as indicated by the difference between Figure 2.4. and Figure 2.5. In Figure 2.4., we have generalized ensemble creation into a broader component called the Postprocessing component. Many systems use postprocessing, although they do not always label it as such. This is because many previously researched systems incorporate more than just ensemble creation as a final modification to the classifier.

Postprocessing in AutoML systems may involve not only combining multiple classifiers through ensemble methods but also refining predictions using techniques such as calibration or error correction. Techniques like model stacking, where the predictions of multiple classifiers are used as features for a meta-model, can further enhance the accuracy of the final classifier by capturing higher-order interactions between base classifiers.

Additional "overhead" steps are typically performed at this stage in the pipeline as well. For instance, if one-hot encoding occurs during preprocessing, decoding must also occur at the end. This process, often considered an extension of preprocessing methods, has not been formally referred to as a separate step. Thus, incorporating it as an additional component is a logical choice.

2.5.1 AutoDL

The winning systems of past AutoML competitions often exhibited relatively low predictive accuracy compared to modern ML systems. Table 2.4. presents the performance of team submissions across the five rounds of the 2017/2018 CHALearn AutoML competition. Due to this comparatively lower accuracy, the focus of AutoML has shifted from handling arbitrary datasets to addressing specific problem domains.

Round	Best Teams	Performance
1	1. KaseyM	0.57
	2. wlWangl	0.56
	3. aad_freidburg	0.55
2	1. wlWangl	0.49
	2. narnars0	0.29
	3. vn.sju	0.28
3	1. aad_freiburg	0.39
	2. narnars0	0.37
	3. hehe	0.34
4	1. aad_freiburg	0.26
	2. thanhdng	0.26
	3. Joy	0.26
5	1. narnars0	0.69
	2. Malik	0.68
	3. wlWangl	0.68

Table 2.4: The results of the 2015/2016 CHALearn AutoML competition [Guy+19]. The competition consisted of 5 rounds, the first four consisting of multiple phases and the final round being only one phase. The performances listed for rounds 1-4 are the performances of each team after the final phase. Since round 5 contained only a single phase the performance values listed for round 5 are from the singular phase. The listed team names are those provided by the individual teams.

AutoDL focuses on automating the creation of deep learning classifiers using neural networks, making it a specific subset of AutoML. While both fields aim to automate classifier development, AutoDL specializes in tasks where deep learning techniques excel. Thus, AutoDL is constrained to a narrower problem space within the broader CASH (Configuration, Algorithm, Selection, and Hyperparameter tuning) problem.

AutoDL is further divided into sub-fields, such as Automated Natural Language Processing (AutoNLP), Automated Computer Vision (AutoCV), and Automated Speech Recognition (AutoSpeech) [Liu+20b][HZC21]. These sub-fields focus on data with inherent relational structures. For instance, images have a specific pixel arrangement, NLP involves grammatical structures, and sensor data typically involves time-series with temporal dependencies.

Though AutoDL systems are a type of AutoML system, they differ from traditional AutoML systems, such as Auto-Sklearn, primarily in their Classifier Selection component. AutoDL systems typically limit classifier options to neural networks, narrowing the range of preprocessing methods to those compatible with deep learning [GBC16]. Consequently, AutoDL systems usually search through fewer states during optimization, which can lead to higher accuracy when applied to datasets with relational features.

AutoDL systems often incorporate domain-specific knowledge or heuristics to optimize preprocessing. For example, data augmentation introduces variations of images or text that maintain conceptual equivalence but differ in appearance. Transfer learning is another technique employed, where a pre-trained classifier is adapted and fine-tuned for a new but related task [TS10]. This leverages pre-existing knowledge from the source task to enhance performance on the target task.

The five top-performing submissions to the 2019 ChaLearn AutoDL Challenge—DeepWisdom, DeepBlueAI, Inspur_AutoDL, PASA_NJU, and Frozenmad [Liu+20a]—utilized a common architecture, illustrated in Figure 2.6. Liu et al. [Liu+21] generalized these systems into a "Global AutoML workflow," which, despite not being universally applicable, provides valuable insight into common AutoDL practices.

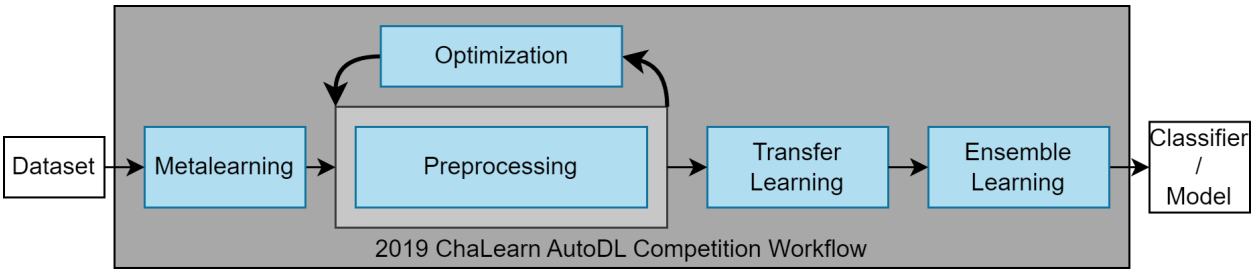

Figure 2.6: The "Global AutoML workflow" as described by Liu et al. [Liu+21]. In this framework, an additional component, Transfer Learning, is placed before ensemble creation. Since the idea of AutoDL is predicated on relational data such as text, images, etc the strict use of only neural networks has, in turn, necessitated the need for an additional component specific to neural networks.

In the framework presented by Liu et al. [Liu+21], a component not found in traditional AutoML systems is the Transfer Learning component. Although transfer learning is not unique to neural networks [LG07], it is a method of warmstarting that is heavily utilized in neural networks [TS10].

2.5.2 Knowledge-Enhanced ML

The use of Knowledge-Enhanced ML for creating predictive classifiers is a widely studied topic across many fields [Cha20] [Sve+20] [HL23]. Research about the theory of enhancing ML with additional knowledge is also vast with methods such as enhancing input data [Bha+20], model selection [Cao+21], or parameter setting [DS19]. The underlying principle of Knowledge-Enhanced ML is that by supplying knowledge to an ML system, the system can use that knowledge to shortcut learning. For example, by using predetermined rules, an ML system may be able to use those rules to ignore certain examples within a training dataset. This may be because there exists some rule, case, etc. for determining a class label in certain scenarios that is known ahead of time, and does not need to be learned.

Providing an ML system with additional knowledge can be implemented by indicating to the ML system certain types of knowledge the system does not need to spend time learning. It can also work by providing an ML system with knowledge that is known to

not exist in a given dataset. Without this missing knowledge, a system would have no way of understanding certain concepts. The more detailed the provided knowledge is, the less learning an ML system needs to perform to attain desirable results. This is further emphasized by the growing sentiment among researchers that current trends in ML are too narrowly focused on neural networks, and that more focus should be placed on symbolic representation, and how to enhance ML with additional knowledge [CM21].

Von Reuden et al. outline a taxonomy that describes the ways knowledge can be integrated into ML systems [Von+21]. They describe their taxonomy using three dimensions: knowledge source, knowledge representation, and knowledge integration. In the context of ML systems, developers and domain experts often provide the primary source of knowledge.

Regarding the second dimension outlined by Von Reuden et al., developers can represent the knowledge they incorporate in many ways. For example, logical rules, probabilities, trees, graphs, etc. structure knowledge representations that developers commonly add to Knowledge-Enhanced ML systems. This diversity in representation highlights the flexibility of incorporating different types of knowledge into ML systems.

The third dimension outlined by Von Reuden et al. determines how the instantiation of an ML system directly incorporates knowledge. For example, the input dataset represents knowledge from which a learner attempts to learn. Alternatively, developers can incorporate it into the classifier through the specification of hyperparameters or the deliberate selection of a specific learner. This taxonomy of knowledge enhancement demonstrates the flexibility and extent to which ML systems can benefit from additional knowledge.

2.5.3 PSML

PSML is a subset of ML that focuses on developing highly specialized ML systems tailored to solve specific individual tasks or problems [BB13][Moo+19][Pła+16]. Unlike more general-purpose ML systems, which aim to perform well across a wide range of tasks, PSML systems aim to produce classifiers that excel at one particular task by leveraging domain-specific

knowledge incorporated during development. At runtime, this additional knowledge can speed up the learning process and/or improve the accuracy of the produced classifier.

Unlike AutoML systems that use a single dataset to learn a classifier for that dataset, PSML systems use a dataset to produce a classifier for multiple datasets. However, the datasets to be classified must still adhere to the schema of the supplied training dataset; otherwise, the learned knowledge structure becomes ineffective for classifying examples.

Classifiers created by PSML systems often perform better than more generalized classifiers for various reasons. First and foremost, PSML systems focus significantly on a specific problem or task. This focused approach allows developers to concentrate their efforts on optimizing a PSML system's ability to produce a classifier for that particular task.

PSML systems often require collaboration between machine learning experts and domain experts. Domain experts provide valuable insights and knowledge about the task at hand, which developers incorporate into the system design and feature engineering process. This integration of domain expertise becomes crucial for developing effective PSML systems. For example, PSML systems rely on customized features specifically tailored to the problem domain. Developers engineer these features based on domain knowledge or extract them from a dataset through preprocessing techniques. Customized features help the system capture relevant information and improve the classifier's performance on a specific task.

PSML systems further utilize specialized techniques well-suited to a problem domain. Since a PSML system focuses solely on a singular task, developers and domain experts can identify niche approaches for learning that work well in isolated scenarios. By using only specialized learners, PSML systems remain constrained to a specific task but can exploit specific behavior unique to that task. The system can then leverage this behavior to improve the performance of the created classifier.

A downside of PSML is that due to the specialization of the learning task, PSML classifiers often perform poorly on problems outside their intended domain. The domain knowledge incorporated into a PSML system during development becomes essentially useless when

confronting problems that cannot use that domain knowledge. For instance, if you give a computer vision classifier text and ask it to predict the subject of a sentence, the classifier would neither understand the semantics of natural language nor interpret the encoding used to represent the text. Without additional development work, PSML classifiers can only be used for the specific problem they were created for.

In relation to Knowledge-Enhanced ML, PSML can be viewed as an extreme instance. Both Knowledge-Enhanced ML and PSML embed knowledge into the final system. However, PSML relies so heavily on domain knowledge that it becomes difficult to use a PSML classifier to make accurate predictions for anything other than the task it has been trained on.

In contrast, Knowledge-Enhanced ML describes a category of ML approaches that aim to utilize knowledge within a system in a more general sense. For example, a classifier trained to classify the famous MNIST digits dataset [GH95] results from a PSML system as it can only output 0" through 9" as possible predictions. In contrast, a system like TransE [Bor+13], which attempts to learn how best to embed concepts into knowledge graphs and relate arbitrary knowledge via translation embeddings, aligns more closely with Knowledge-Enhanced ML.

Chapter 3

Ontology-Enhanced AutoML

In this chapter, we present our main contributions. We start by arguing, in Section 3.1. that ML can be viewed as a spectrum of approaches based on their generality. We then introduce, in Section 3.2. what we call Ontology-Enhanced AutoML, which represents a significant area within the spectrum that remains uncovered. In our presentation of Ontology-Enhanced AutoML, we offer a formal definition, introduce an initial framework, and provide the first instance of this framework for real-world applications.

3.1 Our Vision: The ML Spectrum

In this section, we discuss our vision to conceptualize a spectrum across ML, from AutoML to PSML. A naive breakdown of ML variants might focus on the degree to which a system can automate the process of producing a classifier. However, this categorization results in more exceptions than rules. Fundamentally, any ML system can produce a classifier automatically because learning algorithms, by their very nature, compute results automatically.

The confusion becomes clearer when considering that both PSML and traditional AutoML systems automatically create a classifier from a given dataset. In both cases, after a user provides a dataset, the system performs the necessary computations to create a classifier. So, what distinguishes the two approaches?

The key difference lies in the ability to automate the production of a classifier across a broad range of tasks. For instance, a PSML system might struggle to create a classifier from a dataset that is far removed from its intended domain.

AutoML, on the other hand, can produce a classifier for any dataset regardless of the domain. Although the accuracy may be lower than desirable, a traditional AutoML system can systematically instantiate multiple ML systems to maximize the predictive accuracy of the created classifier.

The extent to which an ML system can work on arbitrary datasets depends on the amount of domain knowledge incorporated into it. This makes intuitive sense: automating classifier creation for a specific domain requires understanding the knowledge associated with that domain. In AutoML, a given dataset could pertain to any domain, so a traditional AutoML system cannot have preconceived notions about the task to be learned.

PSML systems, however, can create highly accurate classifiers for specific domains due to the substantial domain knowledge built into them. While this allows PSML systems to generate accurate classifiers, it also makes it challenging for them to create classifiers that work outside their intended domain.

In ML, the efficacy of learning hinges on a comprehensive understanding of the problem domain. Integrating relevant knowledge helps ML systems make informed decisions about relevant feature associations. Conversely, a lack of relevant knowledge may lead to undesired behavior, where learned knowledge appears beneficial but lacks meaningful correlation to desired outcomes. Therefore, the depth of understanding inherent in the learning process is crucial for mitigating undesirable behaviors.

Thus, the spectrum from AutoML to PSML is differentiated by how much domain knowledge an ML system incorporates. This difference affects an ML system's ability to automate classifier creation across different domains. The ability to automate classifier creation across domains ultimately depends on the amount of domain knowledge present in the system.

On one extreme of the spectrum lies PSML. PSML systems are the least automated ap-

proach in creating classifiers across multiple domains. When a new classifier is needed for a new domain, human interaction is necessary to adapt a PSML system to create the classifier. Often, multiple individuals proficient in various domains are involved. As previously mentioned, domain experts and ML experts frequently collaborate to produce a system capable of creating accurate classifiers. Specialized input encoding, feature extraction, and inference mechanisms are implemented to incorporate as much knowledge into a PSML system as possible. Infusing knowledge into a PSML system is human-intensive and essential for producing accurate classifiers.

On the other extreme is AutoML. Traditional AutoML systems create classifiers fully automatically without human intervention aside from specifying an input dataset. To create a classifier for an arbitrary dataset, a traditional AutoML system must infer as much knowledge as possible from the dataset at runtime. Achieving this is challenging. By using measures from information theory, statistics, and other fields, a traditional AutoML system systematically infers appropriate ML system instantiations to maximize the accuracy of the final classifier.

Close to AutoML is AutoDL. While AutoDL is a relatively generalized approach to ML, it is more specialized than AutoML. By restricting its problem space to accept only correctly formatted datasets, an AutoDL system ultimately cannot produce a classifier for an arbitrary dataset. Instead, an AutoDL system can create more accurate classifiers, but only for a limited subset of problems. Many datasets that traditional AutoML systems can handle are beyond the scope of an AutoDL system. This trade-off of generality is intentional, allowing an AutoDL system to achieve greater accuracy than comparable traditional AutoML systems. However, this trade-off moves AutoDL slightly away from AutoML on the ML spectrum.

On the other side of the spectrum, close to PSML, is Knowledge-Enhanced ML. While Knowledge-Enhanced ML is more specialized than both AutoML and AutoDL, it remains more general than PSML. Knowledge-Enhanced ML systems leverage incorporated knowledge to gain the requisite understanding of the problem space, enabling them to create

classifiers with relatively high accuracy. Figure 3.1. provides a visual representation of the ML spectrum.

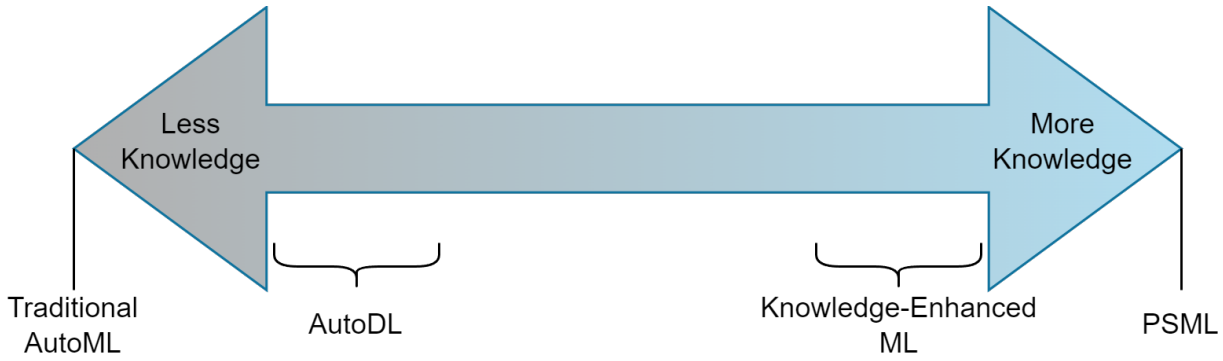

Figure 3.1: The ML spectrum. Along the spectrum lie different approaches to ML. On the end of the spectrum utilizing no domain knowledge during the creation of a model is AutoML. At the other end is PSML which utilizes as much domain knowledge as possible. Between them is AutoDL, and Knowledge-Enhanced.

In the previous discussion on the ML spectrum, we frequently mentioned accuracy. It's crucial to emphasize that accuracy does not determine an approach's position on the spectrum. For example, even if a traditional AutoML system could create a classifier for any domain and achieve perfect accuracy, it would still be classified as a traditional AutoML system. Similarly, a PSML system that, despite having extensive domain knowledge, only produces classifiers with relatively low accuracy, would remain a PSML system.

However, accuracy is significant in evaluating the practical utility of each approach. A traditional AutoML system capable of creating classifiers for any domain would be less valuable if those classifiers had poor accuracy. In practice, researchers and developers prioritize the final performance of the classifiers. Hence, while accuracy is not the defining criterion for an ML system's place on the spectrum, it is a crucial factor in assessing different ML systems.

Research on existing systems shows trends where traditional AutoML systems tend to be less accurate than AutoDL systems, which are generally less accurate than Knowledge-Enhanced ML systems, and so on. These trends highlight the practical considerations that

influence the effectiveness of each approach.

While accuracy does not define the type of ML system, it remains the most critical factor in real-world applications. The accuracy of a classifier directly impacts decision-making, with more accurate predictions leading to better decisions.

Therefore, although accuracy is not the metric for categorizing ML systems, it is of paramount importance. An effective ML system should be able to create classifiers across a wide range of domains and ensure those classifiers are accurate. This leads us to our next section, where we discuss a type of ML system designed to leverage domain knowledge at runtime, enhancing both the accuracy and learning speed of classifiers. Our approach enables the creation of specific system instantiations that, while not containing significant domain knowledge themselves, still utilize domain knowledge to produce accurate classifiers.

We envision a framework that utilizes ontologies to position ML systems between PSML and AutoML on the spectrum. We propose that allowing users to incorporate domain knowledge created by experts into an ML system at runtime can lead to more accurate classifiers than those produced by traditional AutoML systems. We will present our proposed approach in the following section.

3.2 Ontology-Enhanced AutoML

In this section, we introduce an instantiation of our vision: Ontology-Enhanced AutoML. At its core, Ontology-Enhanced AutoML is an ML approach that enables users to integrate domain knowledge by supplying an ontology to the system. We begin by providing a formal definition of Ontology-Enhanced AutoML. Next, we describe our general framework for Ontology-Enhanced AutoML. We then present the initial implementation of this framework, which utilizes a geometric ontology. This instantiation will be employed in our experiments in subsequent chapters to demonstrate the promising potential of Ontology-Enhanced AutoML systems.

3.2.1 Ontology-Enhanced AutoML Definition

So far, we have discussed AutoML as a method for addressing the CASH problem and presented a general framework for creating traditional AutoML systems. However, we have not yet provided a formal definition of a traditional AutoML system. Since Ontology-Enhanced AutoML builds upon AutoML, it is essential first to define the components of a traditional AutoML system. This foundational understanding will allow us to extend AutoML and define what constitutes an Ontology-Enhanced AutoML system.

Definition 3.1. ***Traditional AutoML System***

A traditional AutoML system is a 6-tuple
$(meta, pre_{data}, pre_{feat}, sel_{classifier}, opt, post)$ where

- $Meta$ is the set of all metalearning components a traditional AutoML system could implement, and $meta \in Meta$.
- Pre_{data} is the set of all data preprocessing components a traditional AutoML system could implement, and $pre_{data} \in Pre_{data}$.
- Pre_{feat} is the set of all feature preprocessing components a traditional AutoML system could implement, and $pre_{feat} \in Pre_{feat}$.
- $Sel_{classifier}$ is the set of all classifier selection components a traditional AutoML system could implement, and $sel_{classifier} \in Sel_{classifier}$.
- Opt is the set of all optimization components a traditional AutoML could implement, and $opt \in Opt$
- $Post$ is the set of all postprocessing components a traditional AutoML system could implement and $post \in Post$.

□

From this definition, we understand that a traditional AutoML system consists of a choice for each of the six components outlined in 2.5. With this foundation, we can now define an Ontology-Enhanced AutoML system.

Definition 3.2. ***Ontology − Enhanced AutoML System***

Let $\mathcal{O}$ be the set of all ontologies, and δ be the set of all datasets.

An Ontology-Enhanced AutoML system is an 8-tuple
$(Meta, Pre_{data}, Pre_{feat}, Sel_{classifier}, Opt, Post, Trans, Incorp)$ where

- $Meta$, Pre_{data}, Pre_{feat}, $Sel_{classifier}$, Opt, and $Post$ are the same as in a traditional AutoML system.
- $Trans : \delta \times \mathcal{O} \rightarrow \delta$ is a function that translates a dataset into one that is more conducive to faster and more accurate learning
- $Incorp = \{inc_{Meta}, inc_{Pre_{data}}, inc_{Pre_{feat}}, inc_{Sel_{classifier}}, inc_{Opt}, inc_{Post}\}$
- $inc_{Meta} : \mathcal{O} \times Meta \rightarrow Meta$ is a function that, given an ontology and metalearning component, creates a new metalearning component that is more suitable for the task.
- $inc_{Pre_{data}} : \mathcal{O} \times Pre_{data} \rightarrow Pre_{data}$ is a function that, given an ontology and data preprocessing component, creates a new data preprocessing component that is more suitable for the task.
- $inc_{Pre_{feat}} : \mathcal{O} \times Pre_{feat} \rightarrow Pre_{feat}$ is a function that, given an ontology and feature preprocessing component, creates a new feature preprocessing component that is more suitable for the task.
- $inc_{Sel_{classifier}} : \mathcal{O} \times Sel_{classifier} \rightarrow Sel_{classifier}$ is a function that, given an ontology and classifier selection component, creates a new classifier selection component that is more suitable for the task.

- $inc_{Opt} : \mathcal{O} \times Opt \rightarrow Opt$ is a function that, given an ontology and optimization component, creates a new optimization component that is more suitable for the task.
- $inc_{Post} : \mathcal{O} \times Post \rightarrow Post$ is a function that, given an ontology and postprocessing component, creates a new postprocessing component that is more suitable for the task.

□

One thing to note is that we have discarded the notation for differentiating the training and testing datasets. This is merely for brevity. It is assumed that, following the discussion of CASH in Section 2.2, it is clear that splitting $\mathcal{D}$ into $\mathcal{D}_{train}$, and $\mathcal{D}_{test}$ remains necessary to effectively measure the misclassification rate of an Ontology-Enhanced AutoML system.

3.2.2 A First Framework

Using this definition we can discuss a framework for Ontology-Enhanced AutoML. To begin, for any given Ontology-Enhanced AutoML system, a developer must include an ontology. The ontology included represents the domain knowledge of the desired task. In practice, a user can provide an ontology to an Ontology-Enhanced AutoML system. Alternatively, a creator might build an Ontology-Enhanced AutoML system with a specific ontology built-in, thus limiting the capabilities of the Ontology-Enhanced AutoML system to a specific domain.

$Trans$ and $Incorp$ are two components: the Translation and Incorporation. These two components allow an Ontology-Enhanced AutoML system to utilize an ontology and take responsibility for utilizing the concepts and relationships defined in an ontology. By utilizing the concepts and relationships of an ontology, an Ontology-Enhanced AutoML system can make better-informed decisions about how to create a final classifier.

To use in Ontology-Enhanced AutoML, developers must further adapt the generalized AutoML framework to account for $Trans$ and $Incorp$. To achieve this, they add an additional Translation component before they provide a dataset to the remaining components.

Additionally, they provide an Incorporation component with a given ontology, which they then use to modify the other components of an Ontology-Enhanced AutoML system. Figure 3.2. shows how they incorporate this.

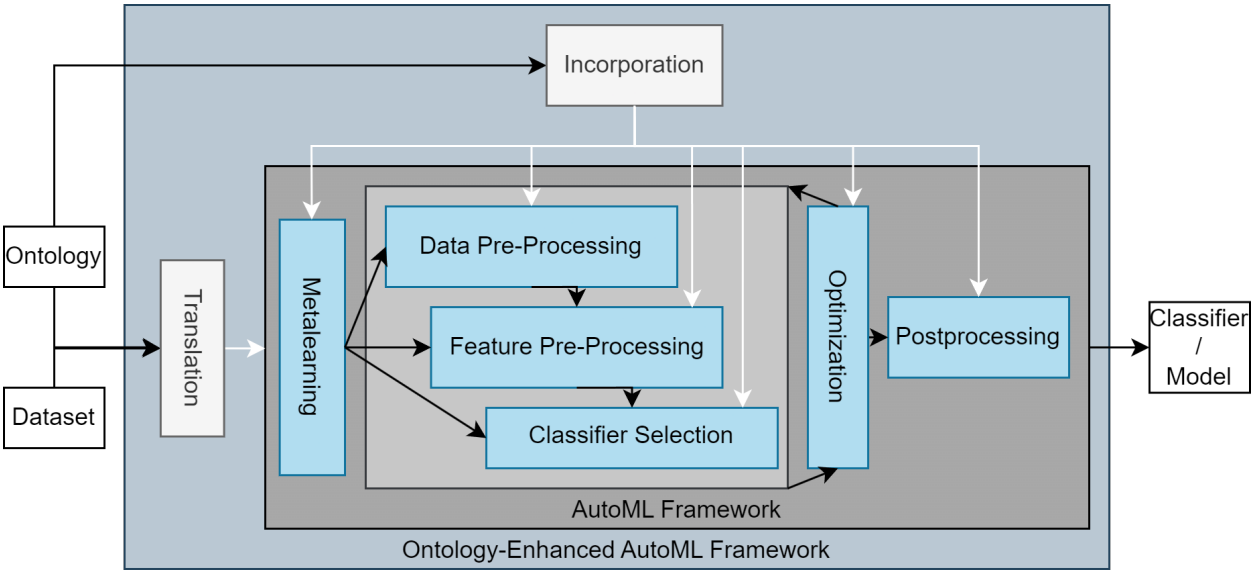

Figure 3.2: Our Ontology-Enhanced AutoML framework contains the additional Translation, and Incorporation component.

The Translation component functions by taking a dataset and an ontology and transforming the dataset into a more suitable representation. By transforming the dataset, an Ontology-Enhanced AutoML system can use a dataset in a format better suited for learning. Ideally, the transformation applied to a dataset would make the knowledge in the dataset easier for a learner to grasp. By making implicit knowledge explicit, for example, the Translation component helps the Ontology-Enhanced AutoML system reduce its computational requirements.

Rather than inferring some knowledge from a dataset through learning, an ontology can provide an Ontology-Enhanced AutoML system with direct access to that knowledge. By matching a dataset to a concept in the supplied ontology, the Translation component uses knowledge from the $Additional_k$ set of the concept.

When the system matches a dataset to a concept, it triggers the execution of $Additional_k$,

causing the Translation component to interpret and use the knowledge from $Additional_k$ to transform the dataset. The system incorporates the knowledge computed by $Additional_k$ into the dataset, resulting in the transformation. For example, if $Additional_k$ specifies that certain feature-values are invalid, the Translation component replaces those feature-values with a value from $Additional_k$ or removes examples with invalid feature-values altogether.

The Incorporation component directs an Ontology-Enhanced AutoML system on how to best utilize the components of a traditional AutoML system. It consists of multiple functions, each corresponding to a traditional AutoML component. For each traditional AutoML component, the Incorporation component's function uses the supplied ontology to create a new, better-suited component.

For every decision an Ontology-Enhanced AutoML system makes, the Incorporation component enables better-informed decisions. By including a function for every AutoML component, the Incorporation component allows the system to use knowledge from a given ontology at every step in the AutoML pipeline.

An Ontology-Enhanced AutoML system can perform similarly to an AutoDL system, for example. By using the Incorporation component, the system can be directed, through an engineered ontology, to expect images as input. This direction comes from creating a concept in the ontology that includes image features. When the Ontology-Enhanced AutoML system matches the dataset to the concept functions in $Additional_k$, the functions are triggered. *Incorp* interprets these triggers to indicate that the system should consider only specific classifiers or use particular feature preprocessing methods, essentially transitioning the system to an AutoDL system.

The exact functionality of the Translation and Incorporation components distinguishes any Ontology-Enhanced AutoML system. Systems with more complex functionality move closer to PSML on the ML spectrum, while systems with simpler functionality align more with AutoML. This range allows Ontology-Enhanced AutoML to cover a broad part of the ML spectrum.

To work effectively, the Translation and Incorporation components must understand a given ontology. Therefore, the user must ensure that the ontology is compatible with the Translation component. The same applies to the Incorporation component. If either component does not understand the knowledge in the $Additional_k$ set, the Ontology-Enhanced AutoML system cannot utilize that knowledge. The ontology's knowledge must also be relevant to the learning task.

The structure of knowledge within $Additional_k$ should follow commonly used structures in ontologies. Domain experts should easily adapt existing ontologies to be understood by an Ontology-Enhanced AutoML system. After adapting an ontology to a specific system, the domain expert can share it with users, allowing them to use specific ontologies for particular problems.

The ecosystem of an Ontology-Enhanced AutoML system resembles modern programming languages and frameworks. Popular programming languages have package managers for incorporating others' packages/libraries into user projects. Similarly, Ontology-Enhanced AutoML systems will have an ecosystem where domain experts engineer and release ontologies for specific domains. Users will select a system and an ontology that meet their needs, allowing them to employ Ontology-Enhanced AutoML without being ML or domain experts concerning their datasets.

If we place Ontology-Enhanced AutoML on the ML spectrum, it ranges from AutoML to Knowledge-Enhanced ML. We intentionally defined Ontology-Enhanced AutoML broadly to allow users to leverage domain knowledge without sacrificing accuracy or compute time. Figure 3.3. shows that Ontology-Enhanced AutoML spans a wide spectrum. The degree of knowledge in a system depends on the specificity of the Translation and Incorporation components by the developer. A system can use more or less knowledge based on the additional knowledge in the inputted ontology. This range of generality depends on the desired use cases.

In Figure 3.3., we illustrate that without an ontology, an Ontology-Enhanced AutoML

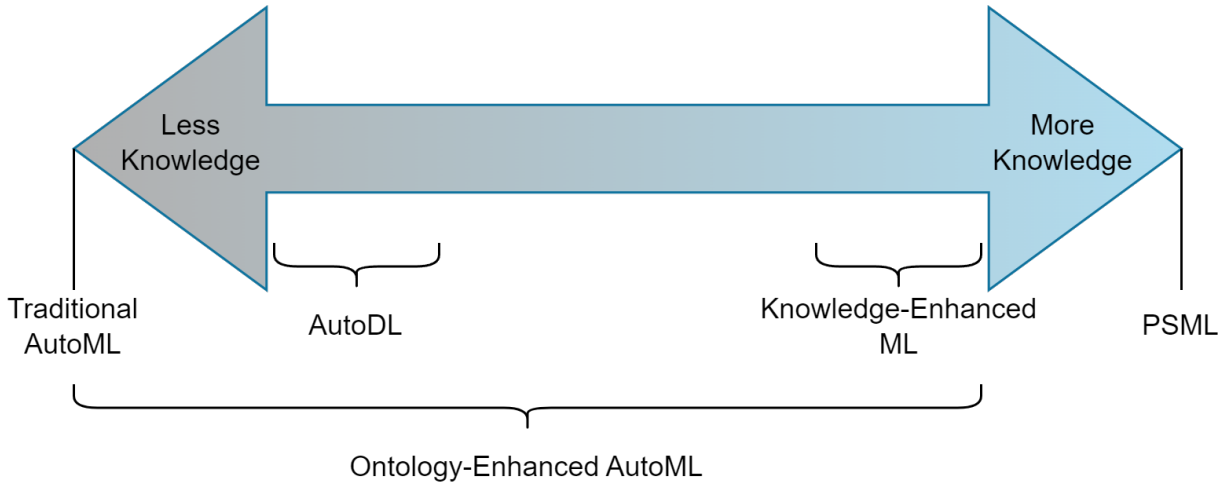

Figure 3.3: The ML spectrum with Ontology-Enhanced AutoML occupying the large gap between AutoDL, and Knowledge-Enhanced ML.

system defaults to a traditional AutoML system. The Translation and Incorporation components use an ontology to modify both the dataset and other components. Apart from these two components, the remaining parts of an Ontology-Enhanced AutoML system mirror a traditional AutoML system. Without an ontology, these two components cannot function, so the Ontology-Enhanced AutoML system uses the traditional AutoML components just like a traditional AutoML system. Similarly, if no concept in a given ontology matches the dataset, the Ontology-Enhanced AutoML system defaults to functioning as a traditional AutoML system.

In such cases, the unmodified dataset essentially bypasses the Translation component. The Incorporation component also cannot modify the traditional AutoML components, and the system operates as a traditional AutoML system. If no domain knowledge is extracted, the Translation component simply returns the input dataset, and each function within the Incorporation component returns the original AutoML component.

In practice, this setup prevents an Ontology-Enhanced AutoML system from failing outright when it does not correctly understand a given ontology. If $Trans$ encounters issues, the Ontology-Enhanced AutoML system may still use $Incorp$ to modify the AutoML compo-

nents, provided that *Incorp* understands the ontology. Likewise, if *Incorp* cannot understand a given ontology but *Trans* can, the Ontology-Enhanced AutoML system will still benefit from the dataset transformation. In the worst case, the system defaults back to a traditional AutoML system, providing redundancy and ensuring that the system can still produce a classifier even without domain knowledge.

The ontology provided to an Ontology-Enhanced AutoML system significantly influences the extent of knowledge the system can utilize. This impact determines where the system falls on the ML spectrum. Incorporating substantial knowledge into both the Translation and Incorporation components moves the Ontology-Enhanced AutoML system closer to PSML. For example, if a developer restricts the system to understanding only domain-specific ontologies, the system will be limited to learning within that domain.

Conversely, Ontology-Enhanced AutoML systems with a broader ability to understand ontologies align more closely with AutoML on the ML spectrum. Ideally, the Translation and Incorporation components would handle a wide variety of ontologies, enhancing the system's versatility and adaptability across different problem domains.

Understanding ontologies broadly bridges the gap between AutoML and PSML on the ML spectrum. Our Ontology-Enhanced AutoML framework offers significant application versatility while retaining specificity for particular tasks or domains. A more generally applicable ontology supports diverse fields, and the framework's ability to cater to multiple fields using the same ontology is a notable advantage.

This adaptability is crucial because it enables the system to operate across various fields without significant modifications. In Ontology-Enhanced AutoML, a system with a general knowledge base can handle different data types and learning tasks without needing substantial structural changes. This versatility allows users to leverage the system for various purposes, applying it across diverse industries and research domains without compromising performance or efficiency. Specialized ML systems may excel in specific tasks but struggle outside their intended scope.

An Ontology-Enhanced AutoML system with broad ontology understanding mitigates the risk of learning outside specific domains. Such a system can efficiently address a wide range of problems, reducing manual intervention or customization. This efficiency is vital in AutoML, where the goal is to automate the ML process as much as possible. By leveraging general ontologies, Ontology-Enhanced AutoML systems streamline model selection, optimization, and deployment across different domains, ultimately saving time and resources.

3.2.3 Ontological Reasoning Mechanisms

Here we present two simple mechanisms that you can implement into both the Translation component and the Incorporation component of an Ontology-Enhanced AutoML system. We introduce these mechanisms to provide concrete examples of how these components can work effectively within Ontology-Enhanced AutoML. We offer two distinct mechanisms: one for performing feature extraction and another for filtering out knowledge that is already known and does not need to be learned by a classifier.

In Section 2.3.3, we briefly discussed how additional knowledge in an ontology helps specify functions used to traverse relationships between concepts. More specifically, we described how the $Additional_k$ set associated with each concept contains the knowledge used to indicate specific functions. The knowledge in $Additional_k$ plays a crucial role in Ontology-Enhanced AutoML by helping understand the relationships presented in a dataset.

$Additional_k$ also relates to the "schema" of a dataset, which we introduced in Section 2.2. As mentioned, the schema provides knowledge such as the units associated with numerical features or the possible feature values a feature may have. For example, consider a dataset about vehicles. Cars might be available in any color, but a dataset representing a specific set of cars might only include cars that are red or yellow. Examining the dataset alone might lead a system to conclude that cars can only be red or yellow, even though they can actually be any color.

Using the *is-a* relationship, an Ontology-Enhanced AutoML system can generate more

generalized knowledge about the underlying concept associated with the dataset. With the knowledge in $Additional_k$, the system can, for instance, use the *is-a* relationship to determine whether to incorporate features from a related concept into the input dataset.

In this way, $Additional_k$ functions similarly to the ABox [Baa03] used in description logics. The ABox represents assertional knowledge—knowledge specific to individual concepts within a domain. Since assertional knowledge is specific to individual concepts, it allows incorporating domain-specific knowledge.

$Additional_k$ acts as a mechanism triggered by matching a dataset to a concept. When a concept with all its features represented in a dataset matches, the system executes the knowledge in $Additional_k$. The formal representation of knowledge in the specified structures within $Additional_k$ is handled functionally, and the procedures dictate the functions performed on the features within a dataset to traverse through the ontology's relationships and derive new knowledge.

By matching a dataset to a concept, the Ontology-Enhanced AutoML system can utilize the triggering of $Additional_k$ to deduce knowledge associated with the matched concept. This approach provides a variable degree of knowledge to the system, allowing it to select which knowledge from $Additional_k$ to use. For example, Figure 3.4. shows how traversing relations between concepts can help "accumulate" knowledge about a dataset.

However, the specific mechanism that performs this traversal must be explicitly defined by the Translation and Incorporation components of an Ontology-Enhanced AutoML system. For our Ontology-Enhanced AutoML framework, we have defined two mechanisms for ontological reasoning within the $Additional_k$ knowledge associated with each concept.

For a given domain, however broad it may be, it is possible to identify specific types of information relevant to the domain which is expected to reside within $Additional_k$. $Additional_k$ can be specified in such a way as to require specific structures such as mechanisms for utilizing the knowledge in an ontology. Additionally, how this knowledge can be utilized depends entirely on how an instantiation may need the knowledge to be represented for a classifier

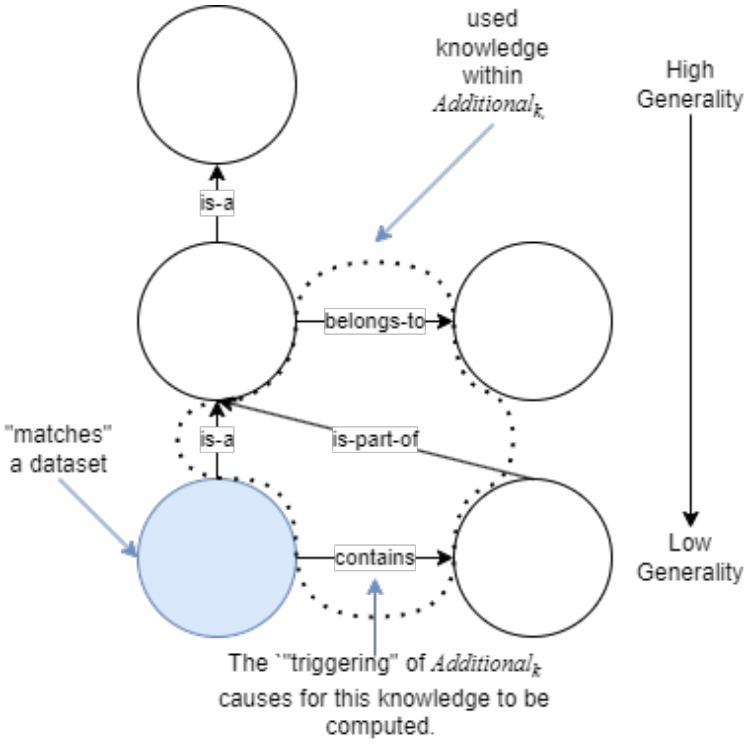

Figure 3.4: How $Additional_k$ can be used to generate knowledge using a provided ontology. The relationships extending a concept can be traversed procedurally to utilize additional knowledge. After matching a given dataset to a concept, $Additional_k$ is "triggered," causing knowledge about specific relationships to be deduced.

to be produced by an Ontology-Enhanced AutoML system.

For our initial framework, the two structured sources of knowledge that we require to be defined within $Additional_k$ are the sets $Extract_k$ and $Label_k$. $Extract_k$ corresponds to the method an Ontology-Enhanced AutoML system can utilize to perform feature extraction on a dataset. To do so, we must extend our definition of concepts which we defined in Section 2.3.3.

Feature Extraction

Before we begin discussing Feature Extraction we must first make a distinction about this mechanism. Our mechanism for performing Feature Extraction is a different mechanism than the feature extraction method used during feature preprocessing. During feature preprocessing, feature extraction takes place by only considering the explicitly defined feature-values

of a dataset. However, there exist scenarios where the feature-values alone are not sufficient for extracting new features.

Definition 3.3. ***Extract$_k$***

Let $Extract_k = \{extract_{k,1}, ..., extract_{k,m}\}$ where $\forall\ extract \in Extract_{k,m}$
$extract : F^+ \rightarrow F_x$ where $extract(v_1, ..., v_n) = fv_{x,y}$ such that $fv_{x,y} \in F_x$ and
$F, F_x \in Feat$. □

This defines $Extract_k$ as a set of functions on the domain of one or more features within $Feat$ that produce a feature value for some other feature also in $Feat$. In essence, these functions take an object and from it produce a feature-value for a feature.

The relationship between the real world and the representation of the real world encoded as a feature is only known by having sufficient domain knowledge of the real world that the feature represents. There is no method in which other features can be utilized, nor any statistical model that can represent that relationship without knowing the domain beforehand.

For example, consider a dataset about engineering design reviews. This dataset contains a list of all individuals who have reviewed the planned construction of a collection of houses. It also contains the past approvals and rejections of specific houses given by the individual reviewers. Now consider giving this dataset to a traditional AutoML system. The system would produce a classifier that would predict the risk associated with approving a house based on who has reviewed it, how big it is etc. However, the produced classifier has no conceptual understanding of the physical object of the house.

Without incorporating some additional knowledge into a given dataset, a traditional AutoML system would produce a classifier that would only be able to make predictions based on factors that have a minor influence on the outcome. As a solution, a feature extraction method for ontological reasoning could be informed of how to extract pertinent features that are directly causal to the predictor class, but that do not exist within the dataset explicitly. A blueprint of each house for example could have features extracted from

it describing the physical layout which, in turn, would have better predictive power when incorporated into the final classifier. The knowledge of a concept *Blueprint* in the set of concepts, and the relationship $hasBlueprint(House, Blueprint)$ in the set of relations could be used to identify the specific blueprints of an example containing a house.

The features that cannot be derived from within a dataset, but are very important to a domain are typically held within the domain knowledge of that domain. There are inherent properties of ontological objects that can only be represented explicitly. If those properties are not expressed as features within a dataset then there is no way to extract that knowledge.

Another example would be a dataset about the sales of a chain of retail stores. Within the dataset is the location of each store, the number of employees, etc. However, if the dataset does not contain information about the actual stock of merchandise each store holds, then it is impossible to gain that knowledge during the learning process of a classifier. What is more likely is that any learned classifier would find less relevant patterns such as the fact that stores with a larger square footage make more sales. While this observation may hold, it is less informative than a classifier that has learned that a specific type of product frequently sells out. The knowledge of a concept $item_1, item_2$ in the set of concepts, and the relationship $bestSeller(item_2)$ in the set of relations could be used to identify the best-selling item in a store.

By utilizing the $Extract_k$ set found within the concepts provided in an ontology, an Ontology-Enhanced AutoML system could perform every function specified within $Extract_k$. For every value returned by one of those functions, an Ontology-Enhanced AutoML system could utilize the value as a feature-value for a new feature. Every example in a given dataset could then be assigned that feature-value for that feature.

A new dataset could then be created using the newly calculated feature-values. This would, in turn, increase the amount of domain knowledge utilized by an Ontology-Enhanced AutoML system. The resulting dataset could then be fed as input into the remaining components of the Ontology-Enhanced AutoML framework. The rest of the components would

then be privy to the domain knowledge specified in the given ontology. This furthermore enables the other components to then make better-informed decisions when creating a final classifier.

Disqualifier/Qualifier

The next ontological reasoning mechanism we have created can be utilized to filter out knowledge from a dataset that is already known before runtime, allowing an Ontology-Enhanced AutoML system to focus on learning other knowledge. The mechanism we have created is called a Disqualifier/Qualifier. The Disqualifier/Qualifier mechanism is a two-part mechanism that is incorporated into both the Translation component, as well as the Incorporation component.

Definition 3.4. ***Label$_k$***

Let $Label_k = \{label_{k,1}, ..., label_{k,n}\}$ where $\forall\ label \in Label_k : F'^{+} \rightarrow F_{label}$
where $label(v'_1, ..., v'_m) = y'$ such that either $y' \in F_{label}$ or $y' = \perp$ and
$F' \in Feat$ □

This defines $Label_k$, similarly to $Extract_k$, as a set of functions on the domain of one or more features within $Feat$. Each function within $Label_k$ produces either $\perp$ or a feature-value for some other feature from $Feat$. However, unlike feature extraction, $Label_k$ is meant to assign a class label for a given example. This becomes useful for attributing class labels to examples that follow some specified pattern, or rule. By returning $\perp$, $Label_k$ signals that the knowledge contained in an ontology was not sufficient to determine a class label for a given example. Thus, the example should be used to learn from.

Figure 3.5. visually demonstrates how the feature extraction and Disqualifier/Qualifier mechanisms of ontological reasoning would fit into our Ontology-Enhanced AutoML framework.

Within the Translation component of an Ontology-Enhanced AutoML system, the disqualifier uses $Label_k$ to generate an associated feature-value of each example. If the feature-

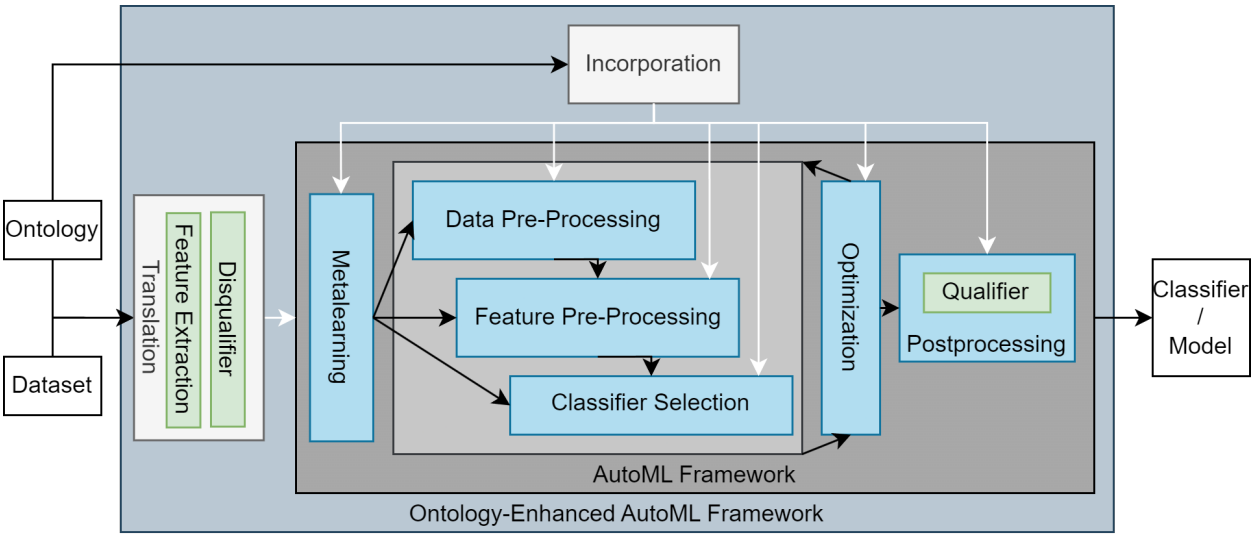

Figure 3.5: Our Ontology-Enhanced AutoML framework with the Disqualifier/Qualifier, and Feature Extraction mechanisms for ontological reasoning included.

value belongs to the set of the predictor feature, that example is removed from the dataset (i.e. disqualified from being learned). If $\perp$ is returned by the Disqualifier then the example remains in the dataset.

The act of disqualifying an example from the dataset is analogous to distilling a dataset by removing examples that do not need to be learned. By leveraging the $Label_k$ set found within a concept matching a given dataset matching, an Ontology-Enhanced AutoML system can filter out all examples where domain knowledge known ahead of time can be used to make a prediction. Because this knowledge can be used to make a prediction, there is usually no need for the example to be used during learning.

For example, imagine a dataset that represents videos. Metrics such as video length, title, description, author, keywords, etc. are contained within the dataset. If the task is to predict what rating a video would get from a collection of viewers then there may exist certain feature-values that can directly inform a prediction. If a large video hosting site already knows that, on average, videos less than two seconds long are given the lowest rating possible then this can be incorporated as a function within the $Label_k$ set of a concept. An Ontology-Enhanced AutoML system could then use this function to identify all videos within a given

dataset that are less than two seconds long and remove them from the training set used during learning. As a result, only relevant examples need to be learned after disqualification has occurred.

In the scenario where a large ratio of the examples contains knowledge that is already known ahead of time, then an Ontology-Enhanced AutoML system would be able to hone in on the resulting examples that remain after disqualification. This, in turn, reduces the amount of computational work the system would need to perform during learning. This would further lead to the Optimization component finishing faster. During optimization, the hyperparameter search space would theoretically be reduced as the Feature Preprocessing, and Classifier Selection components would have much more distilled knowledge to work with. In turn, this would reduce the search space of the Optimization component.

After the disqualifier has been utilized by an Ontology-Enhanced AutoML system to filter out knowledge from a given dataset, the final classifier produced by an Ontology-Enhanced AutoML system would have no understanding of that knowledge. To that end, there exists a second aspect of the mechanism: the Qualifier. The qualifier mechanism ensures that all the knowledge that was removed from a given dataset by the disqualifier is reintegrated back into the final classifier.

By using the $Label_k$ set found within the concept matching a given dataset, the classifier created via the ensemble creation method can be modified to first check a given example against the functions within $Label_k$. If $Label_k$ can produce a feature-value then the given example is predicted to have that feature-value as the predictor class. If $\perp$ is returned by $Label_k$, then the example is passed to the ensemble to be classified. The hand-off of the example from the qualifier to the ensemble acts as a method for ensuring that the knowledge contained within the ontology is present in the produced classifier.

The qualifier piece of the Disqualifier/Qualifier mechanism is incorporated via the $Incorp_{post}$ function of $Incorp$. When an ontology is given to an Ontology-Enhanced AutoML system, $Incorp_{post}$ would take the traditional postprocessing component, and return a modified ver-

sion of it with the qualifier mechanism incorporated into it.

The incorporation of a Disqualifier/Qualifier mechanism is an additional reason we found it necessary to distinguish postprocessing as an additional component in the general AutoML framework. If postprocessing was not an abstracted component then there would be a need to add an additional component within the framework to account for the Disqualifier/Qualifier mechanism. However, this would not work as a general framework since, as stated previously, an Ontology-Enhanced AutoML system is not beholden to using specific mechanisms for utilizing an ontology. By creating a separate component for the Disqualifier/Qualifier, that component would be optional and would not be a universal component. This would defeat the purpose of a general framework.

The use of a Disqualifier/Qualifier mechanism for understanding an ontology can be quite versatile. It can be used to incorporate additional knowledge about what a class label should be based on the additional knowledge contained within a given ontology. It can further be used to incorporate methods, such as feature limitations, into an Ontology-Enhanced AutoML system. In Section 2.2.2. we briefly mentioned how some datasets may have encountered an issue during the collection of their values in the real world. In an instance such as this, the Disqualifier/Qualifier mechanism can be utilized to isolate examples where these issues specifically occurred, and disregard them entirely.

The Disqualifier/Qualifier mechanism could also be used to identify outliers in a dataset. If some property of the feature-values of a given dataset is known before runtime, a given ontology could inform an Ontology-Enhanced AutoML system about what properties expected feature-values should have. If an example from a given dataset does not meet these properties, then it can be discarded during training. This way the time required to perform both data preprocessing and optimization is reduced.

Similar to the effect of reducing the time required to perform computation in certain components, a Disqualifier/Qualifier mechanism has the potential to increase the accuracy of the classifier produced by an Ontology-Enhanced AutoML system. If the $Label_k$ set found

within each concept is perfectly accurate then any feature-value returned by one of the functions within $Label_k$ will accurately predict the feature-value for the predictor class.

3.2.4 Instantiation of the Framework: Geometric Knowledge

The first step towards realizing the goal of Ontology-Enhanced AutoML is to produce a proof-of-concept. The largest hurdle in creating a specific Ontology-Enhanced AutoML system was deciding the method by which we could show a proof-of-concept. It is difficult to describe all the possible ways to incorporate knowledge through the use of ontologies. It is even more difficult to additionally describe how relationships between specific knowledge can be utilized. Because of this, we focused on two methods for ontological reasoning on a dataset with an ontology, and how knowledge described in an ontology can be utilized.

The first mechanism we presented was the feature extraction mechanism for ontological reasoning. Using the feature extraction mechanism, we can essentially shortcut a part of the learning process by making new features readily available to an Ontology-Enhanced AutoML system rather than requiring the system to have to learn these features during training. This frees the system from having to learn already known knowledge and allows it to better spend its resources learning unknown knowledge. The use of knowledge to create additional features is a relatively simple idea, specifically so when considering more generalized areas for applying the Ontology-Enhanced AutoML system. With just a little bit of knowledge, we can increase the accuracy of AutoML systems in areas where previous attempts ultimately defer to using PSML models for improved accuracy.

The use of feature extraction can take many forms within an AutoML system. What is required, though, is that the features required as arguments for a function within $Extract_k$ are present within a given dataset. If a given dataset does not contain any features that are required for the extraction of new features then this essentially indicates to a system that the inputted dataset does not pertain to the domain indicated by an ontology. In such a case, the Ontology-Enhanced AutoML system can fall back to proceed as a traditional AutoML

system. If a dataset includes the necessary features defined by a given ontology, additional knowledge can be extracted for a specific domain, reducing the amount of learning required.

For instantiating our general idea of creating additional features, we chose to go with 3D models and features commonly used in computational geometry. To be more precise, we consider 3D models that are specifically meant to exist in the real world. For these types of 3D models there are certain properties (i.e., physical laws) that are very useful for detecting errors, such as the fact that non-manifold objects cannot exist physically. Figure 3.6 demonstrates the distinction between objects that obey physical laws, and objects that can only be represented computationally. These are specific types of features that can be straightforwardly computed by modern modelling software programs such as Blender [Com18] and therefore can be easily represented in an ontology about 3D objects.

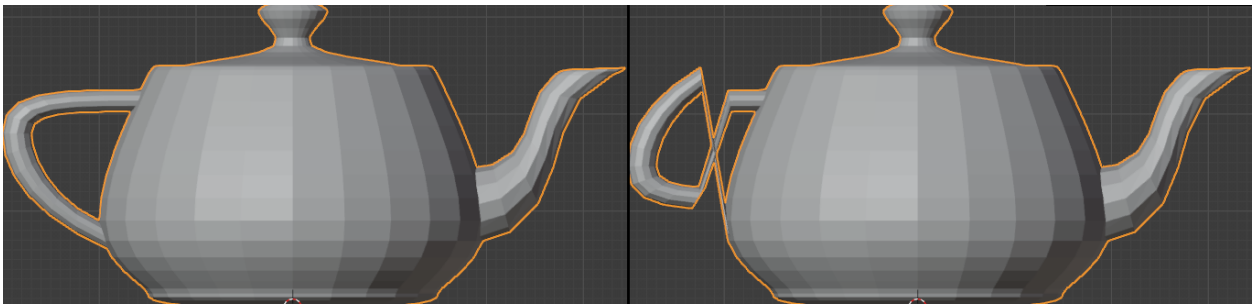

Figure 3.6: An example of a geometry that can be produced physically alongside a geometry that cannot. The model on the right is both non-manifold, as well as contains self-intersecting faces which make it impossible to exist in the real world without modification or approximation.

An additional reason for utilizing 3D models is the numerous advantages 3D modelling provides to a wide range of fields. In academia, 3D modeling research provides a robust framework for studying spatial structures, offering high-fidelity simulations that can be used to visualize complex data or phenomena that are otherwise difficult to observe directly [Gro16]. For instance, 3D models allow engineers to prototype and test virtual designs, saving resources by reducing the need for physical prototypes. In medical fields, 3D models enable practitioners and researchers to simulate human anatomy accurately, providing

insights into surgical planning, biomechanics, and the development of prosthetics [Büc+17]. Similarly, archaeologists use 3D models to reconstruct ancient sites and artifacts, preserving and studying structures that may no longer be intact [BM12].

The issue of poor geometry quality extends outside open-source communities as well. Within the field of archaeology and medicine, for example, researchers have discussed how critical it is to improve the training of individuals responsible for generating geometry [Her23]. A sentiment among researchers is that the use of machine learning may be a potential fix for these issues [Ghi+20], specifically by automatically determining that issues may exist with a 3D model.

The specific ontology we created represents knowledge about the features associated with 3D models. If a dataset contains a 3D model, then there are certain other features that can be extracted from that dataset to provide an Ontology-Enhanced AutoML system with more knowledge about the task to learn. We chose to experiment with 3D models because of a large amount of research showing how certain features of 3D models are useful to many fields [WR16] [Kar+17] [CW19] [Sar+22] [AMD22].

The underlying context of being restricted to Euclidean space makes certain knowledge universal among 3D models. Moreover, additional context may also be present within encoded geometric structures as there are strict physical limitations that allow for only a small number of possible structures capable of achieving specific results (ex. the cube-squared law prohibiting the size of structures [All14]). The extractable features we described in our ontology are as follows:

Number of Non-Manifold Edges

Edges in a 3D model are connections between two infinitely thin faces. If an edge connects to only one face, a hole exists in the model. Similarly, having more than two connecting faces also makes an edge non-manifold.

Number of Bad Contiguous Edges

In computational geometry, when representing a 3D model as a mesh, normals are used to

indicate which direction a face points, thus indicating what is the exterior and interior of an object. If a face's normal points opposite of its neighbours this indicates that that face inverts the interior and exterior of the model at that point.

Number of Intersecting Faces

If two or more faces intersect this indicates that the exterior and interior of the model overlap.

Number of 0-Edges

The number of edges that have 0 length.

Number of Non-Flat Faces

A Non-flat face indicates curvature of a surface that is not explicitly stated in the computational representation and so is ambiguous.

Figure 3.7. illustrates how a model may contain self-intersecting faces. Figure 3.8. demonstrates how an object may appear if it has either non-manifold edges, or bad contiguous edges. Figure 3.9. illustrates an object containing a non-flat face.

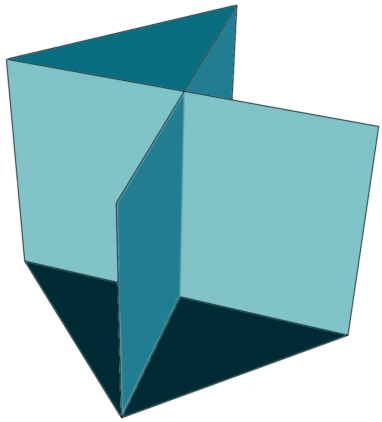

Figure 3.7: An object which contains self-intersecting faces. Although it may appear that this object could exist physically, the faces and edges represent boundaries which separate the interior and exterior of a volume. The intersecting faces makes an interior and an exterior ambiguous

In this specific implementation, it is easy to see how multiple separate domains can utilize the same ontology. Construction and 3D printing, for example, are both able to utilize this

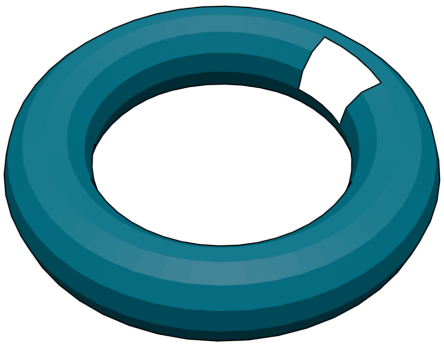

Figure 3.8: An object which contains a hole. We depict a torus to show the difference in the term hole from topology, and computational geometry. We are able to distinguish the interior from the exterior of a torus. However, the hole in the surface of the object could be due to a non-manifold edge, or due to a bad contiguous edge. Rendering software can be optimized to not render faces which are considered interior to a surface and so the hole appears to exist through the entirety of the torus. In reality, only the topmost face is distorted.

ontology as they utilize many applications that require the use of a 3D model. An Ontology-Enhanced AutoML system provided with an ontology containing the above features would be able to extract this knowledge from a dataset, as long as a 3D model is provided.

With this knowledge in mind, we implemented our Ontology-Enhanced AutoML system to specifically utilize the knowledge of the above features. Our system can extract the features described above by using an ontology that pertains to 3D objects. A dataset given to our Ontology-Enhanced AutoML system could be drawn from a variety of domains, however, if it contains features pertaining to a 3D model it is usable by our system.

In other words, the domains to which we limit our Ontology-Enhanced AutoML system are any domains that involve a geometric representation of a real-world object. Construction, manufacturing, architecture, etc. are all fields that may incorporate a form of computation geometry to achieve specific goals. Any domain that similarly utilizes computation geometry could utilize our Ontology-Enhanced system.

The incorporation of a specific Disqualifier/Qualifier mechanism specific to 3D models will allow for our Ontology-Enhanced AutoML system to utilize knowledge pertaining to real-world physics. This will allow our Ontology-Enhanced AutoML system to bypass having to learn certain kinds of knowledge at runtime. Within the real world, the constraints of physics

Figure 3.9: An object which contains a non-flat face. In this instance, a face is composed of four vertices creating a plane. Because any three vertices are co-planar, the fourth vertex creates ambiguity regarding the topology of the face. IT could be that the face "curves" downward, or it could be that it "folds" downward toward.

apply to all geometric objects, thus, we can make predictions about those objects in isolation. If a given dataset contains geometric objects then there is a large amount of knowledge about the laws of physics that can be leveraged to make accurate predictions about the examples. For example, the cube-square law is a significant factor when designing physical aerospace systems [Wer+23][FD03].

The specific Disqualifier/Qualifier mechanism for ontological reasoning that we have utilized is a simple one. The Disqualifier/Qualifier we have implemented simply checks if a 3D model is non-manifold, among a few other properties we discuss later, and if so assigns the example a class specified by the $Label_k$ within a given ontology. Depending on the learning task at hand this is utilized to indicate which specific examples within the dataset involve computational geometry that have no real-world counterpart. The ability to distinguish between examples that have a real-world counterpart and examples that do not helps to reduce the time required for learning. For example, if a classifier is being trained to predict the cost of constructing a skyscraper, none of the other features in an example would matter if the skyscraper fundamentally cannot exist due to its geometry.

When creating an ontology for representing geometric knowledge we have drawn heavy inspiration from Perzylo et al. [Per+15] who have outlined an ontology specifically for

computer-aided design (CAD) data. The ontology Perzylo et al. have created represents geometric constraints of computation geometry in a b-reps representation outlined in Section 2.4. The ontology is described using OWL [AH09]. We have slightly modified the ontology for our use cases as we are only concerned with relatively high-level concepts associated with geometry.

Our modified b-reps ontology specifies the geometric entity concepts *vertices*, *edges*, and *faces*. A vertex represents a vector that corresponds to a 3D coordinate point. An edge represents a line and consists of two vertices. A face represents a surface and consists of three or more edges. Thus a point, an edge, and a surface all have the is-a relationship with a geometric entity that is contained by a mesh. Figure 3.10 depicts the ontology of geometric entities.

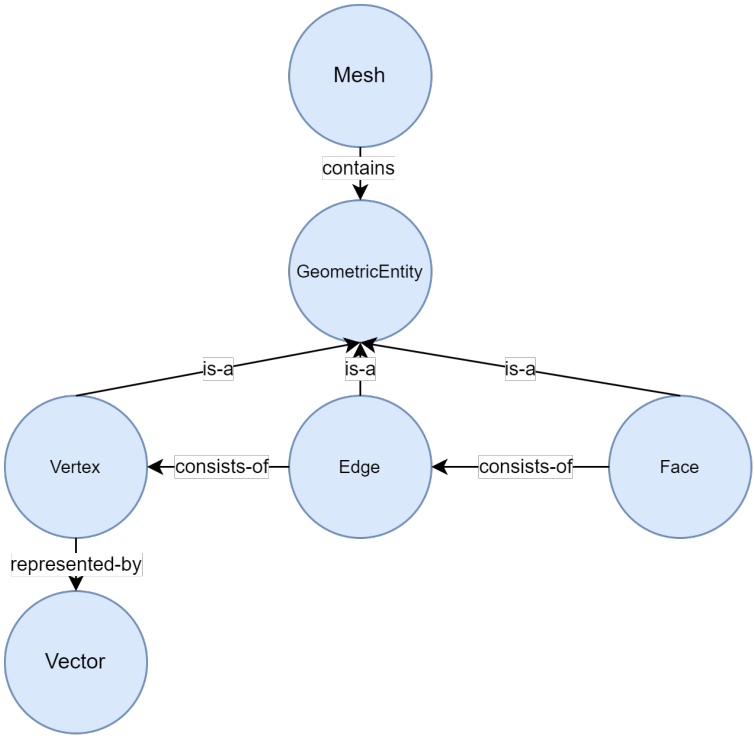

Figure 3.10: The ontology of geometric entities inspired by OntoBrep [Per+15].

From this ontology, by defining the $Label_k$, and $Extract_k$ structures within $Additional_k$ we can specify the features and relations regarding the concepts of a mesh. Regarding the features, the simplest way to represent a mesh with computation geometry is with a list of

vertices. Typically, the faces of a mesh are additionally represented computationally to save on computational resources, however, the only requirement is a list of floats.

To define these as features we first define a vector. Let $V = \{(a, b, c)|a, b, c \in \mathbb{R}\}$ be the set of all vectors. $Vertices$ is the feature of vectors (in computational geometry these are called vertices which is why the vertices are represented by vectors in the ontology) such that $Vertices = \{(v_1, ..., v_n)|\forall i \leq n, v_i \in V, n \in \mathbb{N}\}$. Thus, the feature-values for $Vertices$ are an ordered list of vectors.

The feature-values for edges are tuples of vertices corresponding to the vertices that an edge consists of. Let $Edges = \{(v_{e,1}, v_{e,2})|v_{e,1}, v_{e,2} \in V\}$ be the feature associated with edges. Since a face consists of three edges we let $Faces = \{(e_1, e_2, e_3)|e_1, e_2, e_3 \in Edges\}$ be the feature associated with faces.

Table 3.1. outlines the features associated with each concept.

Concept	Feature
Vector	V
Edge	$Edges$
Face	$Faces$
Mesh	$Vertices$

Table 3.1: The features associated with each concept.

The order of the vectors is important because it encodes the edges, and the faces of a 3D object implicitly. Every three vectors is considered to be a triangle with edges between each of them. Representing a 3D object this way simplifies the structure, however, there will exist duplicate vertices.

Alongside these features, we specify the following $Extract_k$, and $Label_k$ sets of each concept.

- **Mesh**
  - $Extract_{mesh} = \{extract_{nonManifoldEdges}, extract_{badContiguousEdges}, extract_{intersectingFaces}, extract_{zeroFaces}, extract_{zeroEdges}, extract_{nonFlatFaces}\}$ where
    $extract_{nonManifoldEdges} : Vertices \rightarrow nonManifoldEdges$

$extract_{badContiguousEdges} : Vertices \rightarrow badContiguousEdges$

$extract_{intersectingFaces} : Vertices \rightarrow intersectingFaces$

$extract_{zeroFaces} : Vertices \rightarrow zeroFaces$

$extract_{zeroEdges} : Vertices \rightarrow zeroEdges$

$extract_{nonFlatFaces} : Vertices \rightarrow nonFlatFaces$

and where the $nonManifoldEdges$, $badContiguousEdges$, $intersectingFaces$, $zeroFaces$, $zeroEdges$, $nonFlatFaces$ sets are features composed from the set of integers.

- $Label_{mesh} = \{label_{manifold} : Vertices \rightarrow \{true, false, \perp\}\}$

- **Face**
  - $Extract_{face} = \{extract_{area}, extract_{normal}\}$ where

    $extract_{area} : Faces \rightarrow Areas$

    $extract_{normal} : Faces \rightarrow Normals$

    and where $Areas$ is a feature composed from the set of reals and $Normals$ is a feature composed from V.
  - $Label_{face} = \emptyset$

- **Edge**
  - $Extract_{edge} = \{extract_{length} : Edges \rightarrow Lengths\}$

    where $Lengths$ is a feature composed from the set of reals.
  - $Label_{edge} = \emptyset$

The matching of a dataset to the *Mesh* concept can trigger the computation for the extraction of the number of non-manifold edges, the number of bad contiguous edges, the number of intersecting faces, the number of 0-edges, and the number of non-flat faces are extracted. The matching of the *Face* concept can trigger the computation for the extraction of the area and normal of a face. The matching of the *Edge* concept can trigger the computation for the extraction of the length of an edge.

As we have specified, a mesh has the extracted features *nonManifoldEdges*, *badContiguousEdges*, *intersectingFaces*, *zeroFaces*, *zeroEdges*, and *nonFlatFaces* which we have yet to discuss. These properties pertain to the desired features outlined above. To compute the values for these properties we must further add constraints to a geometric object represented by our ontology. These are as follows:

1. ManifoldEdgeConstraint: This constraint is defined between an edge and two faces. The placing of this constraint specifies that an edge must be constrained by two faces. If an edge does not adhere to this constraint then it is considered non-manifold. To be constrained by two faces means that both faces consist of the edge, and the constraint is not held.

2. NormalConstraint: This constraint applies to the relative orientation of normal vectors between two faces and is bi-directional. For the constraint to be held, the normal vectors of both faces must not be directly opposite each other. Specifically, if the dot product of the normal vectors is negative, indicating that the vectors are facing in roughly opposite directions, the constraint is not satisfied. However, special consideration is needed for sharp edges, where this criterion might not apply directly due to geometric discontinuities or the specific nature of edge constraints (Blender [Com18] applies this constraint to edges where the angle between adjacent faces is greater than a threshold).

3. NoIntersectionConstraint: This constraint is defined between two faces and is bi-directional. Firstly, each face must be triangulated, then for each triangle in each face it is compared to each triangle in the other face using a triangle-triangle intersection algorithm [TTS06]. If any triangles intersect, the constraint does not hold.

4. NoFaceConstraint: This constraint is defined on a single face. If the area of a face is 0 this constraint is not held.

5. NoEdgeConstraint: This constraint is defined on a single edge. If the length of an edge is 0, this constraint is not held.

6. NonCollinearityConstraint: This constraint is defined between a face and three vertices. Firstly, for all edges the face consists of, if none of the edges consist of one of the vertices in the constraint then this constraint does not hold. Secondly, if the cross product of any two of the points representing the vertices is not 0, then the constraint does not hold. Finally, if the cross product of the remaining point with any of the previous two points is not 0, then the constraint does not hold.

To determine if these constraints are held for a given 3D model some computation must be performed. To perform these calculations, each vertex is represented by a vector that contains three properties: x, y, and z. Each of these is a double which, when interpreted together, specifies a coordinate in 3D space. To calculate the length of an edge the distance between the two vertices that bound the edge can be calculated. Once calculated, the *length* property can be added to the edge. Similarly, an *area*, and *normal* property can be added to a face by calculating the triangular area. The precise calculation involves passing the edge lengths of all the edges bounding the face to Heron's formula [Wei].

The $contains(Mesh, GeometricEntity^{+})$ relationship denotes the entirety of a mesh. A mesh consists of multiple faces, edges, and vertices and so all geometric entities present in a 3D object must reside in this relationship. $is\text{-}a(Face, GeometricEntity)$, $is\text{-}a(Edge, GeometricEntity)$, and $is\text{-}a(Vertex, GeometricEntity)$ are relationships used to specify the specific faces, edges, and vertices a mesh is comprised of.

The relationships $consists\text{-}of(Edge, Vertex^{2})$, and $consists\text{-}of(Face, Edge^{3})$, represent the "binding" of vertices to edges and faces. If an edge consists of two vertices, and a face consists of three edges then a face also consists of three vertices. This relationship is the knowledge of how to construct a geometric object from the $Vertices$ feature of a dataset.

Lastly, the relationship $is\text{-}represented\text{-}by(Vertex, Vector)$ is a method to join the numerical values of a feature-value of $Vertices$ into the concepts of a mesh. At the lowest level,

the feature of the dataset that gets mapped to a concept is fundamentally done using this relationship.

By interpreting the $contains(Mesh, GeometricEntity^{+})$ relationship defined in the geometric modelling ontology, an Ontology-Enhanced AutoML system would understand that a mesh must contain a geometric entity. When presented with an example containing a feature-value from $Vertices$, the definition for the ontology can be used to match the dataset to the associated concepts of the ontology. If no geometric entity is contained within, then the example does not contain a valid geometric object. Similarly, if a mesh contains less than four faces then it is also not able to represent a valid geometric object. We know this to be true as if there are less than three faces, it is impossible to construct a solid that does not break at least one of the constraints defined above.

To help with understanding our ontology, we present an example of a 3D object represented by our ontology. Figure 3.11. depicts the geometric object we are creating using the ontology.

The mesh (simply named *Object*) contains a set of geometric entities.

```
Individual: ex:Object
    Types:
        ex:Mesh
    Facts:
        ex:contains                 ex:Face1
        ...
        ex:contains                 ex:Face4
```

From the visual we can see that there exist four faces, each of which is contained within the geometric object. We can then proceed with instancing the faces.

```
Individual: ex:Face1
    Types:
        ex:Face
    Facts:
        ex:consistsOf    (ex:Edge1 ex:Edge11 ex:Edge12 ex:Edge13)
```

Here we make use of the RDF listing structure to provide order of the edges in the

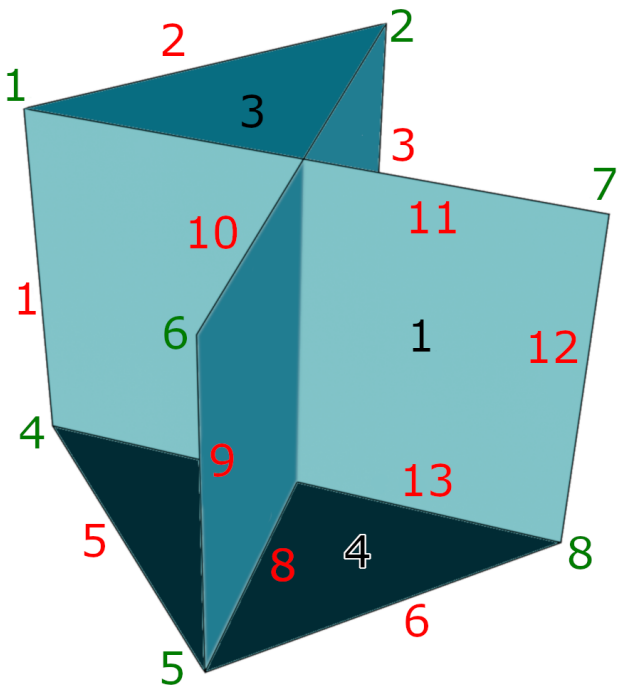

Figure 3.11: The geometric object we are trying to represent using our ontology. The black lines represent explicit edges. Black numbers are used to indicate faces. Red numbers are used to indicate edges. Green numbers are used to indicate vertices.

consistsOf relationship. This is important because the normal of a face is determined by the order of the bounding edges.

```
Individual: ex:Edge1
    Types:
        ex:Edge
    Facts:
        ex:consistsOf     (ex:Vertex4 ex:Vertex1)
```

Finally, we can represent vertices and vectors

```
Individual: ex:Vertex4
    Types:
        ex:Vertex
    Facts:
        ex:representedBy   ex:Vector2
```

```
Individual: ex:Vector2
    Types:
        ex:Vector
    Facts:
```

```
        ex:x    "-0.5"xsd:double
        ex:y    "-0.5"xsd:double
        ex:z    "-0.5"xsd:double
```

For each of these objects, it is possible to iterate over them and to construct a set of constraints such as *ManifoldEdgeConstraint(Edge1, Face1, Face3)* which detail the specific constraints a geometric model must contain. After all constraints are constructed the process of creating the desired features involves iterating over all geometric entities and ensuring that it maintains all constraints that apply to it. For example, all edges should have a ManifoldEdgeConstraint. If an edge does not have this constraint then we increment the number of non-manifold edges by 1. At the end of the calculation, the property *nonManifoldEdges* of the mesh object can be appended as a new feature to a given dataset. After all geometric entities are iterated over, the resulting properties computed by $Extract_{mesh}$ are used by the Translation component of the Ontology-Enhanced AutoML system to create a new dataset with these features appended.

From the object instances created using this ontology, an Ontology-Enhanced AutoML system can use the $Label_{mesh}$, and $Extract_{mesh}$ sets that are associated with a *Mesh* concept. Within $Extract_{mesh}$ we define the facts related to a mesh as the new features that can be extracted using the ontology. These are the *number of non-manifold edges*, the *number of bad contiguous edges*, the *number of intersecting faces*, the *number of 0 faces*, the *number of 0 edges*, and the *number of non-flat faces*. We can further specify that any mesh that contains any non-manifold edges is immediately known to have no real-world representation. To that end, we specify within $Label_{mesh}$ that the *Mesh* concept should return a false when given as input.

The exact method of performing these computations requires defining the implementation for the functions within $Extract_k$ and $Label_k$. In application these functions are easily computed by many modern 3D modelling programs such as Blender [Com18]. OWL, however, is not Turing-complete and cannot perform arbitrary computations. Semantic Web Rule Language (SWRL) can be used to extend OWL to allow for arbitrary computation.

SWRL contains mechanisms called "built-ins" (custom functions for performing more complicated behaviour) that can be written as rules within a concept. SWRL can be integrated to allow the engineer of an ontology to implement "built-ins" that invoke functions within an associated program. When interpreting the ontology, an Ontology-Enhanced AutoML system would reference the rules specified by SWRL and invoke the function defined within by utilizing the associated program.

The incorporation of SWRL into an Ontology-Enhanced AutoML system allows for arbitrary computation, however, it also requires incorporating a reasoner (the underlying mechanism for reasoning about the defined SWRL rules within an ontology). As discussed before, including a mechanism for deducing an ontology sometimes makes a system undecidable. A developer of an Ontology-Enhanced AutoML system can make it so that the only rules that will be referenced by an Ontology-Enhanced AutoML system are those that adhere to a specific format. The format specified by a developer should be designed to require that a given ontology does not require any reasoning and that the "built-ins" simply execute any associated functions with the proper arguments.

Alternatively, it is also possible for a developer to specify a structure for naming properties that can be used by an Ontology-Enhanced AutoML system to parse the name of a function associated with a concept. An associated program would then be required that the system could reference for executing the function. For example, a system could require that a property with the form $function_{<}name>$ where $<name>$ is used to specify the name of the function to invoke. Any object that matches the concept containing this property can be given as input.

For both of these options, the developer of an Ontology-Enhanced AutoML system would need to specify how external libraries can be loaded. For example, when using SWRL, the developer could specify what specific SWRL reasoner is used. The developer could then require all "built-ins" to include an accompanying Python implementation that can be hot-loaded at runtime to be used to execute the "built-ins" functions.

The developer of an Ontology-Enhanced AutoML system can also dictate other, less formalized parameters associated with a system. For example, the developer of an Ontology-Enhanced AutoML system could require that a user provide as input an index or similar mechanism for identifying the feature to extract from. In our specific instantiation, the edges and faces are constructed by iterating over the $Vertices$ feature of a dataset. When a dataset is given as input, our instantiated system looks for the feature named "vertices" and upon finding it, uses that as the mesh.

In this way, the idea of Ontology-Enhanced AutoML becomes quite versatile. Users would be able to select systems with specifications meeting their exact needs as well as ontologies that suit the system. This, in turn, would reduce the time required to produce a classifier, and likely increase the classifier's accuracy as well.

Now that we have demonstrated a first instance of our Ontology-Enhanced AutoML framework we can proceed to present our evaluation of it by experiments.

Chapter 4

Experimental Evaluation

In this chapter we present the experiments we performed to demonstrate the abilities of Ontology-Enhanced AutoML. We first discuss the specific datasets we used in our tests. We then outline the experimental setup we used. Following this we discuss the results of the experiments.

To show what we performed, each experiment using the same Ontology-Enhanced AutoML system, albeit with modifications to the $Label_{mesh}$ set associated with the *Mesh* concept created using the supplied ontology.

4.1 Datasets

In this section we present the datasets we used for our experiments. We used two datasets created for previous research in computational geometry, the ABC dataset [Koc+19], and the Thingi10k dataset [ZJ16]. Both datasets contain a large quantity of examples to train with, and more importantly, contain more features than just 3D models which can be used in the base implementation of our experiments for use in training.

The ABC dataset [Koc+19] is provided as a collection of 100 batches each containing 10,000 geometric models with associated files such as statistics, features, meta information etc. Of these associated files, the statistics file is important for our experiment as it contains

the values for the features we used for learning and can be found in A.1. Within the statistics files there exist features such as the number of edges of the associated geometry, the number of vertices of the geometry, etc. However, there also existed features such as curves that were represented as a list of the different types of curves within the geometry. Not only is the list's length dependent on the geometry, and so is not consistent in length across geometry, but it also primarily contains many repeated values. For example, a sample curve value for one of the models is '[Circle, Circle, Circle, BSpline, Circle, BSpline, Circle, ...]'. Normally these features would be included in an ML system, however, the base Auto-Sklearn system cannot directly deal with these features without some additional manual preprocessing by a user. Since this is not desirable for the base system we opted to eliminate these features. The specific features we excluded from the experiment were the curves and surfs features.

During experimentation, we used only the first two batches of files which contained 10,000 models each. We considered that 20,000 models would be sufficiently large to showcase the potential of our framework while also being small enough that it is feasible to work with. However, not all models contained a corresponding statistics file and so the models were filtered to contain only those with an associated statistics file. This left 14,609 models for learning.

The Thingi10k dataset [ZJ16] is a collection of 10,000 models, alongside a table of features similar to the statistics files from the ABC dataset. A difference, however, is that the table of features in the Thingi10k dataset had more than twice the number of features present in the ABC dataset. The features that were part of the ontology were also manually constructed for the Thingi10k dataset. The reasoning given for constructing these features, given by Zhou, and Jacobson [ZJ16], was that mesh quality plays a critical role in 3D model usage "in the wild." Naturally, this produces additional processing work but can show how important this information is. We agree that this knowledge is vital for applications involving 3D models which is why it was included in our ontology. To that end, we wanted to see the difference between having the ontology and not. We also wanted to see if a supplied ontology could

recreate these features, and if doing so would be helpful. Thus, when testing the difference between our Ontology-Enhanced system, and the base system, we removed these features from the Thingi10k dataset. Otherwise, we would not be able to create a valid comparison between the two systems.

4.2 Methodology

In this section we discuss the methodology of our experiments. We first discuss the methods employed when modifying Auto-Sklearn to create an Ontology-Enhanced AutoML system. When the discuss the two separate experiments performed. For each experiment, we tested the ability to modify an AutoML system to show the differences between the inclusion of each mechanism for ontological reasoning.

To begin with, the experiments were run on a single machine with specifications found in Table 4.1. The machine was reformatted prior to the experiments to ensure that minimal software would interfere with or increase the wall-clock time of computation. The machine was left disconnected from any network while performing the experiments to prevent unwanted network software from taking control of the processor.

CPU	Intel i7-12700K
CPU Speed	3.60 GHz
RAM	32 GB
OS	Ubuntu 22.04.2 LTS

Table 4.1: The technical specifications of the machine used for experimentation

4.2.1 Auto-Sklearn Modifications

Auto-Sklearn [Feu+15] is an open-source automated machine learning (AutoML) toolkit built on top of scikit-learn. It automates the process of selecting the optimal machine learning pipeline for a given dataset, including data preprocessing, feature selection, model selection,

and hyperparameter optimization. The codebase of Auto-Sklearn is designed to be modular and extensible, allowing developers to understand, modify, and extend its functionality easily.

At the core of Auto-Sklearn is the Bayesian Optimization mechanism, which is implemented using the SMAC3 (Sequential Model-based Algorithm Configuration) library. The Bayesian optimization process is encapsulated in the `automl_smbo.py` module, where the `AutoMLSMBO` class manages the generation and evaluation of different pipeline configurations. This class orchestrates the optimization loop by proposing new configurations based on previous performance results and updating the surrogate learner to reflect the observed performance landscape.

Auto-Sklearn implements meta-learning to warm-start the optimization process by leveraging prior knowledge from previous experiments on similar datasets. The meta-learning component is handled in the `metalearning` package, particularly in the `meta_features.py` and `meta_learning.py` modules. These modules extract meta-features from the current dataset and retrieve the most promising pipeline configurations from a meta-data repository. By starting the optimization with these configurations, Auto-Sklearn can more quickly instantiate high-performing learners.

The ensemble construction in Auto-Sklearn is a novel component that combines multiple models to improve predictive performance and robustness. The code responsible for building ensembles is located in the `ensemble_builder.py` module. Auto-Sklearn uses an ensemble selection algorithm that iteratively adds classifiers to the ensemble based on their performance on a validation set. The `EnsembleBuilder` class periodically scans the output directory for new classifiers, evaluates them, and updates the ensemble weights accordingly. This process ensures that the final ensemble benefits from the diversity of classifiers explored during the optimization process.

Data preprocessing and feature engineering are integral parts of the machine learning pipeline in Auto-Sklearn. The toolkit defines a range of preprocessing components, such as imputation, encoding, scaling, and dimensionality reduction, in the `preprocessing` package.

Each component adheres to the scikit-learn transformer interface, allowing them to be easily integrated into pipelines. The pipeline assembly is managed by the `AutoSklearnPipeline` class in the `pipeline.py` module, which dynamically constructs pipelines by selecting and sequencing preprocessing steps alongside the estimator.

For hyperparameter optimization, each machine learning algorithm in Auto-Sklearn defines a search space for its hyperparameters. These definitions are provided in the `configspace` objects within the respective algorithm modules found in the `classification` and `regression` packages. The `HyperparameterSearchSpace` class specifies the types of hyperparameters (categorical, ordinal, continuous), their ranges, and default values. During the Bayesian optimization process, the `SMAC` optimizer samples from these search spaces to propose new configurations.

Another complication resulting from the use of Auto-Sklearn is that the Bayesian Optimization mechanism dynamically allocates computational resources based on the user-specified time limit for the system. Auto-Sklearn uses the remaining runtime allocation after each iteration to determine how much time should be allocated for the next iteration of optimization.

Our modification of Auto-Sklearn is implemented via a "wrapper" class which wraps Auto-Sklearn. When a user creates a new instance of the wrapper class, a provided ontology is used to modify and construct an appropriate instance of the Auto-Sklearn class. When instantiating Auto-Sklearn normally users must specify a time budget, which dictates the total allowable time for the optimization process. We removed that parameter from our wrapper. Because we wanted to assess the time it took the Optimization component of our instantiations to converge we minimized the influence the time limit had on the optimization process. Instantiating our wrapper class dictates an 80 hour maximum runtime.

Auto-Sklearn leverages this time budget to manage computational resources and to determine the allocation of effort across different phases of optimization. By default, Auto-Sklearn continues the optimization loop until the specified time limit is exhausted. However,

to observe whether there is a notable increase in performance with respect to time, we implemented an early stopping mechanism within the Auto-Sklearn codebase. The Auto-Sklearn "train" function parameterizes a higher-order function which is invoked every time a classifier is added to the ensemble. With our early-stopping mechanism we assumed that the systems would stop due to our early stopping mechanism long before hitting the 80 maximum runtime limit. By implementing such a large time limit Auto-Sklearn would theoretically allocate resources in such a way that each run was allocated more than enough time to process, and thus, not influence the final performance of the classifiers.

When implementing the early stopping mechanism there were several factors to consider. Because Auto-Sklearn produces an ensemble classifier as a last step there should exist some minimum number of ML systems that must be instantiated before optimization can finish. This is further supported in the documentation for Auto-Sklearn which recommends a minimum of 10 learned classifiers should be created before terminating the optimization process. We adhered to this recommendation and set the early stopping mechanism to wait until at least 10 classifiers had been trained.

We also had to come up with a standard for determining when sufficient learning had occurred. At what point is it decided that a classifier performs *well enough*? The justification we used was that once learning had ceased to cause a noticeable increase in the accuracy of a learned classifier learning was considered finished. As the performance of a classifier increases there comes a point when increases in accuracy see diminishing returns. Continuing to learn after this point runs a risk of overfitting a classifier to a dataset which is also undesirable.

During the optimization process, we measured the accuracy of the best-performing classifier after each iteration. We compared the 10 most accurate classifiers within the current lifetime of the system and averaged the result. If all 10 of the most accurate classifiers had an accuracy within ten percent of their mean we considered the optimization to have converged. The specific choice of ten percent is arbitrary, however, it is small enough that moderate improvements to performance during optimization would not cause early stopping.

A minor complication of early stopping is that Auto-Sklearn does not compute the final ensemble of classifiers until learning stops. Once learning finishes the system uses the performance of each classifier to create a boosted ensemble [Pol12] weighted by overall classifier performance. Because of this, the 10 most accurate classifiers do not necessarily indicate how well the final ensemble will perform. Boosting algorithms boast the mantra of "the whole is greater than the sum," which, in practice, often creates ensembles that outperform any individual classifier. Because of this, we felt justified in our early stopping mechanism. In theory, the final ensemble performance should be greater than that of the mean of the 10 most successful classifiers which does not detract from our experiments. By implementing these two checks for early-stopping we can bypass Auto-Sklearn from running until the time budget specified.

Within our Auto-Sklearn wrapper class we further implemented helper functions for utilizing the ontology to perform feature extraction and disqualification. Our wrapper class expects the ontology to contain two functions which can be given a 3D model as input. The first of these functions corresponds to the $Extract_k$ set defined in Section 3.2.3. A provided dataset is iterated over and each example contained within is given to the $Extract_k$ function which then returns the values defined in Section 3.2.4. Similarly, the second function: $Label_k$ must exist within the ontology to label examples. Within our ontology the labeling of an example further required the extracted features computed by $Extract_k$ and so these were also required by the helper function. In order to determine the feature corresponding to the 3D model of a dataset we exposed a parameter to the user which was used to indicate which feature index in a dataset corresponded to the 3D model.

The last addition to Auto-Sklearn within our wrapper class was the insertion of a Qualifier into the ensemble. We created an additional wrapper class which wrapped the `AutoSklearnClassifier` class representing the trained ensemble created during training. An instance of our Auto-Sklearn Classifier wrapper embeds within itself an instance of the $Label_k$, and $Extract_k$ functions. It then preceeds the prediction step of the AutoSklearn-

Classifier instance by first extracting the features of a 3D model and labelling the 3D model. If $Label_k$ is able to predict a given example then the computed label is returned as a prediction. Otherwise, the example, alongside the newly extracted features is given as input into the Auto-Sklearn classifier to be predicted by the ensemble. Figure 4.1. depicts our implementation. Overall, our wrapper class consists of approximately 200 lines of code, alongside various helper functions totaling approximately 1000 lines of code total.

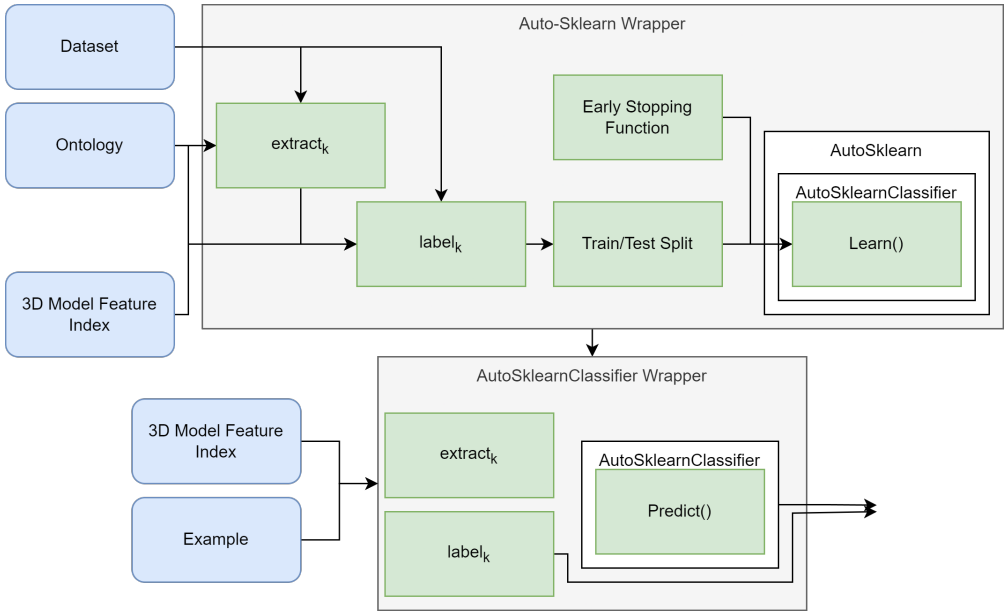

Figure 4.1: A diagram showing our implementation consisting of a wrapper class for Auto-Sklearn, and a wrapper class for the Auto-Sklearn classifier. Inputs are depicted in blue and functions are depicted in green. White components represent the original Auto-Sklearn code-base while gray components represent our extension to Auto-Sklearn.

4.2.2 Experiments

The first system tested was Auto-Sklearn [Feu+15]. Auto-Sklearn is a good option for experimentation considering it is open-source, easy to use, easy to extend, and contains fairly thorough documentation. Auto-Sklearn as a system handles all aspects of an AutoML system including meta-learning, data-preprocessing, feature pre-processing, classifier selection, hyperparameter selection, optimization, and ensemble mechanisms. To that end, there exists much in terms of how to customize Auto-Sklearn for our purposes.

In the second system, we left the $Label_k$ set of the *Mesh* object empty as we wanted to test the Feature Extraction mechanism for ontological reasoning in isolation. By doing this we are essentially creating an Ontology-Enhanced AutoML system that is slightly less general on the ML spectrum than AutoML.

For the third system, we then introduce the rule

```
IF Mesh.nonManifoldEdges > 0 THEN NON-REPRESENTABLE
```

Introducing this rule acts as a way to test the Disqualifier/Qualifier mechanism for ontological reasoning we have described. Furthermore, by introducing this rule our system moves a little bit away from traditional AutoML on the ML spectrum. In conjunction with the second system, our third system essentially takes two discrete "steps" closer to PSML on the ML spectrum. By doing so we show that our framework can be utilized to create systems on multiple points of the spectrum. Each "step" helps to demonstrate that a single Ontology-Enhanced AutoML framework can be instantiated in multiple ways to generate a variety of systems.

There exists, however, an obstacle that we had to overcome before beginning our experiments. Neither of our chosen datasets contained any class labels. As we are learning a classification task a class label must exist to be learned. Additionally, because we are attempting to show the generality of our framework we could not pick an identical classification task for both datasets.

To overcome this obstacle we used an approximation to generate what the class labels should be for the examples of each dataset. Rather than attempting to learn the same knowledge, we wanted to see the extent to which the ontology could be used within other domains of interest that are parallel to 3D printing. For the Thingi10k dataset, we referenced Schneider et al.'s research [Sch+22] which was a previous study on the Thingi10k dataset. In Schneider et al's. [Sch+22] study, Finite Element Analysis (FEA) was performed on each geometry within the Thingi10k dataset. However, as the author mentions, it is impossible to do FEA analysis without some modifications to the geometries under certain conditions.

These conditions are when the model is non-manifold, has self-intersections, or contains gaps in the geometry. Specifically, the rules we created for the Thingi10k dataset class labels were

```
IF Mesh.nonManifoldEdges > 0 OR
IF Mesh.badContiguousEdges > 0 OR
IF Mesh.intersectingFaces > 0 OR
IF Mesh.zeroFaces > 0
THEN NON-FEA-ABLE
```

For the ABC dataset, we manually generated rules specifying the class labels. To achieve this, we referenced Blender's 3D print toolbox [Com18] which is primarily used to help individuals discover defects in 3D models which would likely fail when 3D printed. However, since Blender's 3D print toolbox is the tool we used to extract new features of 3D models (as we discuss later) we used a relaxed set of rules derived from Blender. The relaxed set of rules we used to generate the class labels was

```
IF Mesh.nonManifoldEdges > 0 OR
IF Mesh.badContiguousEdges > 0 OR
IF Mesh.intersectingFaces > 0.1 * |Faces| OR
IF Mesh.zeroFaces > 0.1 * |Faces|
THEN NON-3D-PRINTABLE
```

What these rules imply is that 3D printing is still possible even with minor defects of the faces of a model. Although the boundary representation of 3D models implies that faces with zero area create infinitely thin surfaces, in practice the layers of a 3D printer have a thickness so minor defects involving faces with zero area may still be able to produce a 3D model. It may be more likely that the produced model is inaccurate compared to the encoded computational representation, however, it would still be technically printable. By using the rule described above we were able to separate examples based on whether the geometry was "3D printable." 3D printable appears in quotes here because it was infeasible to attempt to 3D print all 10,000 models and so the rule acts as an approximate for the ability of a model

to be 3D printed.

After class labels had been generated for both datasets we fed each dataset with class labels into all three systems. Following this we also fed the ontology into the second and third systems. Figure 4.2. helps with understanding our experimental setup by illustrating how each experiment was performed.

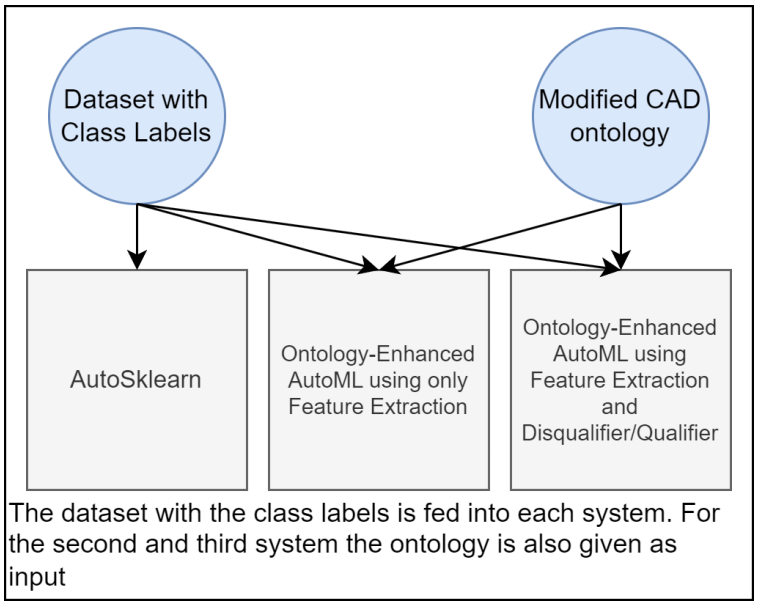

Figure 4.2: The general setup of the experiments. The dataset and ontology are given as input into the appropriate systems.

Another difference between the two experiments is that the Thingi10k dataset contained two geometries that contained over 2 million vertices. One issue with having such a large number of vertices is that the computation time to extract features for geometries increases with the number of vertices. This is because calculating the constraints specified in the ontology for a given model is a lengthy process. Because of this, these two models alone each took over an hour to perform Feature Extraction. Obviously, this is a downside of the system that we will discuss in the next section.

Because we only had two datasets to work with, we decided to split each dataset into sub-datasets similar to how cross validation would occur. Using stratified sampling, we created 10 sub-datasets for both datasets. Each sub-dataset contained 90% of the examples

for training and 10% of the examples for testing. This allowed us to perform the experiment in a more generalized manner which further allowed us to perform more tests.

Since AutoML systems are evaluated based on both their cost, as well as their accuracy we recorded the time taken by a system to produce a classifier, as well as the accuracy of said classifier. However, while Feature Extraction may increase accuracy some overhead comes with doing the computation required to perform the extraction. Thus we also recorded the time taken specifically by the Ontology-Enhanced system to have more detailed information regarding each run.

As a quality of life improvement when testing, for both the system using only the Feature Extraction mechanism, and the system using the additional Disqualifier/Qualifier mechanism we performed Feature Extraction once and shared the resulting feature-values across systems. This was implemented in such a way as to ensure that it did not influence the results of the time required to perform optimization. In the results section this will be apparent when examining the time required to perform feature extraction across the experiments.

We additionally decided to not use an ontology language to represent the ontology described in Chapter 3. The ontology was instead hard-coded into the experiment via predefined $Extract_k$, and $Label_k$ functions for performance reasons. As discussed previously, the feature extraction process is already quite lengthy and adding an additional layer of computational requirements would only exacerbate this issue. Furthermore, the procedural nature of the $Label_k$ and $Extarct_k$ sets means that the process of matching a dataset to a concept, extracting features, and labelling examples would proceed identically regardless of whether the ontology was provided, or if it was hard-coded. We discuss the implications of this in Section 4.4.

One problem, however, exists: how do the translation and incorporation components know how to invoke arbitrary functions specified in the ontology? Furthermore, how do these components know which feature in an example contains the vertices of a mesh? This is the aspect of Ontology-Enhanced AutoML on which a developer of an Ontology-Enhanced

AutoML system has the most influence.

4.3 Results

In this section we present the results of our experiments. The summarized results of the experiment involving the ABC dataset can be found in Table 4.2. In the ABC dataset, we see that the base Auto-Sklearn system was unable to achieve an accuracy greater than 0.90. Conversely, both of our Ontology-Enhanced systems had almost perfect accuracy across all 10 runs, with a minimum accuracy achieved of 0.99 across any run. The average accuracy of the base Auto-Sklearn system was 0.89, and the average accuracy of the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism, and for the Ontology-Enhanced AutoML system containing both mechanisms was 0.99, and 0.99 respectively. This is an increase of 0.09 and 0.10 compared to the base Auto-Sklearn system. In relative terms, this was an increase in accuracy by approximately 11% for both Ontology-Enhanced AutoML systems compared to the base Auto-Sklearn system.

System	Accuracy	Total (s)	Optimization (s)	FE (s)
Auto-Sklearn	0.89	1668	1668	0
FE only	0.99	19804	1689	18115
FE and DQ	0.99	18221	105	18115

Table 4.2: The averaged results of testing the base Auto-Sklearn system, and our Ontology-Enhanced systems on the ABC dataset. FE = Feature Extraction, DQ = Disqualifier/Qualifier

Importantly, we see that the base Auto-Sklearn system performed better with regard to optimization time. The maximum time required to produce a classifier in the base system was 2968 seconds, whereas the minimum time required to produce a classifier in either of our Ontology-Enhanced systems was 17937 seconds. It is worth noting, however, that the time required to produce a classifier in the Ontology-Enhanced AutoML systems is predominantly used by the system to perform Feature Extraction. Across all the Ontology-Enhanced AutoML system runs, the percentage of time spent optimizing was, at worst,

greater than 85%. When comparing *just* the optimization time, the average of all the base system runs is 1668 seconds. The average time for the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism was 1689 seconds. The average time for the Ontology-Enhanced system containing both mechanisms was 105 seconds. This equates to an increase in optimization time of 21 seconds for the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism, and a decrease of 1562 seconds for the Ontology-Enhanced system containing both mechanisms.

If we look at the individual runs of the ABC dataset experiment in Table 4.3. we see that for the base Auto-Sklearn system, the lowest accuracy occurred in run 3 with an accuracy of 0.89. Interestingly, the lowest accuracy for all three systems occurred in different runs with an accuracy of 0.99 in run 6 for the Ontology-Enhanced AutoML system which utilized Feature Extraction only, and an accuracy of 0.99 in run 0 for the Ontology-Enhanced AutoML system using both ontological reasoning mechanisms. There was no single run that performed worst or best in all three systems.

There exists a large spread of values for the optimization time across the base Auto-Sklearn system. Some runs took under 500 seconds, while others took close to 3000 seconds.

The observation that the accuracy gain is consistent across all runs and both systems for the ABC dataset experiment is exhibited in Table 4.4. There we see that the largest difference between any two runs is 0.02 for the Ontology-Enhanced AutoML system which used only Feature Extraction. For the Ontology-Enhanced AutoML systems which used both ontological reasoning mechanisms, the largest difference between runs was 0.01.

The spread of values for optimization time is highlighted in Table 4.5. There we see that the Feature Extraction system is unable to influence the time to optimize in a meaningful manner. In half of the runs, the Feature Extraction system took longer than the base Auto-Sklearn while in the other half, it performed faster. The Feature Extraction and Disqualifier/Qualifier system was able to optimize faster than the base Auto-Sklearn system in all 10 runs. However, the degree of improvement is relatively dispersed. The average gain

Run	Accuracy	Total (s)	Optimization (s)	FE (s)
Auto-Sklearn 0	0.89	2965	2965	0
Auto-Sklearn 1	0.89	2968	2968	0
Auto-Sklearn 2	0.89	382	382	0
Auto-Sklearn 3	0.89	2886	2886	0
Auto-Sklearn 4	0.89	384	384	0
Auto-Sklearn 5	0.89	2978	2978	0
Auto-Sklearn 6	0.89	187	187	0
Auto-Sklearn 7	0.89	380	380	0
Auto-Sklearn 8	0.89	3279	3279	0
Auto-Sklearn 9	0.90	266	266	0
FE 0	0.99	21153	3120	18032
FE 1	0.99	21216	3131	18085
FE 2	0.99	18075	248	17826
FE 3	0.99	21431	3125	18306
FE 4	0.99	18625	343	18282
FE 5	0.99	21291	3100	18190
FE 6	0.99	18307	243	18063
FE 7	0.99	18359	216	18143
FE 8	0.99	21312	3115	18196
FE 9	0.99	18274	245	18029
FE and DQ 0	0.99	18131	98	18032
FE and DQ 1	0.99	18194	109	18085
FE and DQ 2	0.99	17937	111	17826
FE and DQ 3	0.99	18417	111	18306
FE and DQ 4	0.99	18390	107	18282
FE and DQ 5	0.99	18298	108	18190
FE and DQ 6	0.99	18162	99	18063
FE and DQ 7	0.99	18251	108	18143
FE and DQ 8	0.99	18310	114	18196
FE and DQ 9	0.99	18119	90	18029

Table 4.3: The results of testing the base Auto-Sklearn system, and our Ontology-Enhanced systems on the ABC datasets. We performed two experiments, one to measure the results of using the Feature Extraction mechanism, and one to measure the results of additionally using the Disqualifier/Qualifier mechanism. FE = Feature Extraction, DQ = Disqualifier/Qualifier

Run	FE only Accuracy Gain	FE and DQ Accuracy Gain
0	0.09	0.09
1	0.10	0.10
2	0.10	0.10
3	0.11	0.11
4	0.11	0.11
5	0.09	0.09
6	0.09	0.10
7	0.09	0.09
8	0.09	0.09
9	0.09	0.09

Table 4.4: The gain in accuracy across runs for the ABC dataset. The Feature Extraction Accuracy Gain indicates how much accuracy was gained when implementing only the Feature Extraction. The Disqualifier/Qualifier Accuracy Gain column indicates how much accuracy was gained when implementing both ontological reasoning mechanisms.

in optimization was 1562 seconds with a standard deviation of 1349 seconds.

We see the largest improvement to optimization time in the 0^{th} run for the Feature Extraction and Disqualifier/Qualifier system with a value of 2867 seconds. Referencing the Feature Extraction system it is likely that the improvement for this run would be slightly better if the Feature Extraction mechanism was not used. This is because the Feature Extraction system for the 0^{th} run performed worse than the base Auto-Sklearn system by 154 seconds.

The averaged results of the experiment involving the Thingi10k dataset can be found in Table 4.6. In the Thingi10k dataset, we see that the base Auto-Sklearn system performed similarly to the ABC dataset and was unable to achieve an accuracy higher than 0.89. Our proposed Ontology-Enhanced systems achieved an accuracy of 1 for all runs of the experiment except run 4 of the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism. In that run, the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism achieved an accuracy of 0.99. The average for the base Auto-Sklearn system was 0.88. the average accuracy for the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism, and for the Ontology-Enhanced AutoML system containing both mechanisms was 1.00. This is an increase of 0.12 compared

Run	FE only Optimization Gain (s)	FE and DQ Optimization Gain (s)
0	-154	2867
1	-162	2859
2	133	270
3	-239	2774
4	40	276
5	-121	2870
6	-56	87
7	164	271
8	164	3165
9	21	176

Table 4.5: The gain in Optimization time across runs for the ABC dataset. The FE-only Optimization Gain indicates how many seconds were saved during Optimization when implementing only the Feature Extraction. The FE and DQ Optimization Gain column indicates how many seconds were saved during Optimization when implementing both ontological reasoning mechanisms.

to the base Auto-Sklearn system. This accounts for an increase of 13% in accuracy to the base Auto-Sklearn system.

System	Accuracy	Total (s)	Optimization (s)	FE (s)
Auto-Sklearn	0.88	1326	1326	0
FE only	1.00	23280	1271	22009
FE and DQ	1.00	22495	485	22009

Table 4.6: The averaged results of testing the base Auto-Sklearn system, and our Ontology-Enhanced systems on the Thingi10k dataset. FE = Feature Extraction, DQ = Disqualifier/Qualifier

Once again, the Feature Extraction mechanism uses the bulk of the wall-clock time for the Ontology-Enhanced AutoML systems. Across all the Ontology-Enhanced AutoML system runs, the percentage of time spent optimizing was, at worst, greater than 82%. The average time required to produce a classifier in the base system was 1326 seconds. The average optimization time for the Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism was 1271 seconds. The average optimization time for the Ontology-Enhanced system containing both mechanisms was 485 seconds.

If we look at the individual runs of the Thingi10k dataset experiment in Table 4.7. we see that for the base Auto-Sklearn system, the lowest accuracy occurred in run 6 with an

accuracy of 0.88. We also see that only one run achieved an accuracy of less than 1.0 for both Ontology-Enhanced AutoML systems which was run 4 for the Feature Extraction mechanism only system.

The spread-out nature of optimization times was less pronounced for the Thingi10k dataset experiment than was the case for the ABC dataset experiment. For the base Auto-Sklearn system, two of the ten runs have an optimization time that is magnitudes larger than the other eight runs: runs 1 and 3 which achieved an optimization time of 3597 seconds and 3734 seconds respectively. When looking at the Feature Extraction mechanism Ontology-Enhanced AutoML system we see that the disparity in optimization times increases to the point where four of the ten runs are larger by orders of magnitude. However, the spread of optimization times disappears entirely for the Feature Extraction and Disqualifier/Qualifier system which had all ten runs optimized within 36 seconds of one another. Across all runs the Feature Extraction and Disqualifier/Qualifier system performed better than the base Auto-Sklearn system with regards to optimization time. The difference between the fastest Auto-Sklearn run, and the slowest Feature Extraction and Disqualifier/Qualifier system run was 182 seconds.

The spread of values for optimization time is similar to the ABC dataset experiment as is shown in Table 4.9. There we see that the Feature Extraction system is similarly unable to improve the time to optimize in any meaningful manner. In more than half of the runs, the Feature Extraction system took longer than the base Auto-Sklearn while in four runs it performed faster. The Feature Extraction and Disqualifier/Qualifier system was able to optimize faster than the base Auto-Sklearn system in all 10 runs. However, the degree of improvement is also relatively dispersed. The average gain in optimization was 1840 seconds with a standard deviation of 1169 seconds. Although there is an improvement in the optimization time, the degree of improvement is highly disparate.

We see the largest improvement to optimization time in the 1st run for the Feature Extraction and Disqualifier/Qualifier system with a value of 3100 seconds. Similar to the

Run	Accuracy	Total (s)	Optimization (s)	FE (s)
Auto-Sklearn 0	0.89	750	750	0
Auto-Sklearn 1	0.86	3597	3597	0
Auto-Sklearn 2	0.89	713	713	0
Auto-Sklearn 3	0.87	3734	3734	0
Auto-Sklearn 4	0.88	718	718	0
Auto-Sklearn 5	0.88	724	724	0
Auto-Sklearn 6	0.85	735	735	0
Auto-Sklearn 7	0.89	693	693	0
Auto-Sklearn 8	0.88	701	701	0
Auto-Sklearn 9	0.88	895	895	0
FE only 0	1.00	22009	628	21381
FE only 1	1.00	26573	4653	21919
FE only 2	1.00	23624	1336	22287
FE only 3	1.00	22923	733	22189
FE only 4	0.99	22614	818	21796
FE only 5	1.00	23242	866	22376
FE only 6	1.00	23132	1251	21881
FE only 7	1.00	23034	668	22366
FE only 8	1.00	22555	618	21937
FE only 9	1.00	23097	1136	21960
FE and DQ 0	1.00	21893	511	21381
FE and DQ 1	1.00	22416	496	21919
FE and DQ 2	1.00	22775	487	22287
FE and DQ 3	1.00	22671	482	22189
FE and DQ 4	1.00	22279	483	21796
FE and DQ 5	1.00	22855	478	22376
FE and DQ 6	1.00	22357	475	21881
FE and DQ 7	1.00	22847	481	22366
FE and DQ 8	1.00	22417	480	21937
FE and DQ 9	1.00	22440	479	21960

Table 4.7: The results of testing the base Auto-Sklearn system, and our Ontology-Enhanced systems on the Thingi10k dataset. We performed two experiments, one to measure the results of using the FE mechanism, and one to measure the results of additionally using the DQ mechanism.

Run	FE only Accuracy Gain	FE and DQ Accuracy Gain
0	0.11	0.11
1	0.14	0.14
2	0.11	0.11
3	0.13	0.13
4	0.12	0.12
5	0.12	0.12
6	0.15	0.15
7	0.11	0.11
8	0.12	0.12
9	0.12	0.12

Table 4.8: The gain in accuracy across runs for the Thingi10k dataset. The Feature Extraction Accuracy Gain indicates how much accuracy was gained when implementing only the Feature Extraction. The Disqualifier/Qualifier Accuracy Gain column indicates how much accuracy was gained when implementing both ontological reasoning mechanisms.

ABC dataset experiment the improvement for this run would likely be significantly better if the Feature Extraction mechanism was not used. This is because the Feature Extraction system for the 1st run performed worse than the base Auto-Sklearn system by 1056 seconds.

Run	FE only Optimization Gain (s)	FE and DQ Optimization Gain (s)
0	121	238
1	-1056	3100
2	-622	226
3	3001	3252
4	-100	234
5	-141	245
6	-515	259
7	25	212
8	82	220
9	-241	415

Table 4.9: The gain in Optimization time across runs for the Thingi10k dataset. The FE-only Optimization Gain indicates how many seconds were saved during Optimization when implementing only the Feature Extraction. The FE and DQ Optimization Gain column indicates how many seconds were saved during Optimization when implementing both ontological reasoning mechanisms.

In Table 4.10. we show the relative gain in accuracy across all runs between our Ontology-Enhanced AutoML systems, and Auto-Sklearn. Here we see that the Ontology-Enhanced AutoML system with the Feature Extraction mechanism resulted in the greatest increase in

accuracy while the Ontology-Enhanced AutoML system with both mechanisms resulted in basically no effect on accuracy.

Dataset	FE only Accuracy Gain	FE and DQ Accuracy Gain
Thingi10k	0.12	0.12
ABC	0.09	0.10

Table 4.10: The average gain in accuracy across runs for the Thingi10k, and ABC dataset. The FE-only Accuracy Gain indicates how much accuracy was gained when implementing only the Feature Extraction mechanism. The FE and DQ Accuracy Gain column indicates how much accuracy was gained when implementing both ontological reasoning mechanisms.

Conversely, in Table 4.11. we see that the Ontology-Enhanced AutoML system with the Feature Extraction mechanism had a roughly 50% chance of either increasing or decreasing the time required to perform optimization. For the Thingi10k dataset, the Ontology-Enhanced AutoML system with the Feature Extraction mechanism averaged an increase of optimization time by 13.52%. The Ontology-Enhanced AutoML system with both mechanisms averaged a decrease of optimization time by 44.71%.

Dataset	FE only Optimization Gain (s)	FE and DQ Optimization Gain (s)
Thingi10k	55	840
ABC	-21	1417

Table 4.11: The average decrease in time spent optimizing across runs for the Thingi10k, and ABC dataset. The FE-only Optimization Gain indicates how much optimization time decreased when implementing only the Feature Extraction mechanism. The FE and DQ Optimization Gain column indicates how much optimization time decreased when implementing both ontological reasoning mechanisms.

4.4 Discussion

This section will focus on the discussion and interpretation of our results. The results demonstrate that our proof-of-concept regarding system Ontology-Enhanced AutoML is a good first step. First, we discuss the increase in accuracy we observed from the experiments as a result of the Feature Extraction mechanism of our Ontology-Enhanced AutoML system. This will lead us to talk about issues that arose from performing Feature Extraction. Following this

we talk about the improvement to the time required for Optimization when including the Disqualifier/Qualifier mechanism into our Ontology-Enhanced AutoML system. Finally, we discuss some potential concerns we feel may be present in our research.

4.4.1 Increase in Accuracy

In both experiments, we see an increase by approximately 10% in the predictive accuracy of the produced classifier regardless of the ontological reasoning mechanism implemented to that of the base Auto-Sklearn system. This is a good sign as it provides evidence for the utilization of ontologies in AutoML. It furthermore shows the promise of Ontology-Enhanced AutoML.

The base Auto-Sklearn is already able to achieve a relatively high level of accuracy within our experiments compared to scenarios such as the ChaLearn competition. The high accuracy of Auto-Sklearn made it so that there was a smaller window of opportunity for improvement. The level of increase in accuracy could be considered the result of over-fitting, however, we do not understand this to be the case as we address in Section 4.4.4.

We also theorize that the Qualifier/Disqualifier component could potentially perform better with regard to accuracy than we see in these experiments. Because the accuracy of Feature Extraction alone is already near perfect there is little room when additionally adding the Disqualifier/Qualifier mechanism for ontological reasoning.

Naturally, this suggests that an increase in domain knowledge also leads to an increase in accuracy. By allowing for the incorporation of an ontology, Ontology-Enhanced AutoML can utilize the knowledge found within an ontology to increase the accuracy of the produced classifier.

4.4.2 Issues with Feature Extraction

The time required to produce a classifier was substantially less in the base Auto-Sklearn system than in our Ontology-Enhanced systems. Despite the impact on cost, we believe that

these experiments lay the groundwork for future research in the field of AutoML.

Naturally, extracting features has a cost associated with it. The runtime cost of extracting the features was averaged at 18115 seconds and 22015 seconds for the ABC dataset and Thingi10k dataset, respectively. What this would look like in the real world is a front-loading of compute time required before any learning could take place.

Additionally, this front-loading is more substantial in this particular case because of the high cost of extracting the features we specifically indicated in our ontology. Different ontologies would come with their trade-offs between the compute time required to generate certain values associated with the supplied knowledge, and the accuracy required for a particular application. 3D modelling specifically requires a significant amount of data to computationally represent which is only exacerbated by the graph-like structure used in the representation. In this specific domain, the time required to compute these features was substantial.

However, domains where the knowledge contained within a dataset can be extracted in a much more compute-friendly manner would see better benefits of our approach. We theorize that the difference in these two times arises due to two factors: An increased number of features to learn from, and the knowledge being more *distilled* facilitating easier learning. These two factors seem to be at odds with each other, and so future work would be required to determine the associated weight of each factor in determining training time.

For example, imagine a decision tree learner. By introducing more features to the learner it may be that the branching of the nodes happens more effectively due to an increase in information gain. However, it could also be the case that the introduction of new features also makes it so that the decision tree now has to learn further leaf nodes to encompass the newly added features. In one instance the time to train such a decision tree is decreased. In the other instance, the training time is increased.

One of the primary challenges encountered in our specific Ontology-Enhanced AutoML system pertained to the time-consuming nature of Feature Extraction, particularly in the context of computational geometry. Feature Extraction for computational geometry tasks

involves lengthy computations, and the length grows with the complexity of the 3D models being processed. As a result, extracting features from highly detailed or intricate 3D models can significantly increase the time required for this Feature Preprocessing mechanism.

4.4.3 Decrease in the Time to Optimize

Promisingly, the loss in time is somewhat offset by the increase in the performance of the Optimization component. By utilizing our Disqualifier/Qualifier mechanism for ontological reasoning we saw that although the Feature Extraction mechanism could potentially increase the optimization time of a system, the Disqualifier/Qualifier mechanism resulted in an overall lower optimization time. This is specifically highlighted in run 1 in Table 4.9. Here we see that Feature Extraction added close to twenty minutes to the optimization time. However, even with this drastic increase in optimization time, the Disqualifier/Qualifier mechanism was able to reduce the optimization time compared to the base Auto-Sklearn system by 35%.

What this tells us is that different mechanisms for ontological reasoning can improve an Ontology-Enhanced AutoML system in isolated ways. What this entails is that it may be possible to build an Ontology-Enhanced AutoML system by using specific mechanisms to specifically optimize certain outcomes of the system. If a developer wants to prioritize decreasing the time required to optimize a classifier then they could use mechanisms that do just that. Conversely, if a developer prioritized accuracy then they could similarly use mechanisms that specifically increase accuracy.

These results suggest that not only is ML indeed a spectrum but that it is entirely possible to manufacture an Ontology-Enhanced AutoML system in such a way that it can be placed on the ML spectrum wherever is needed (within reason). A user would then be able to select whichever Ontology-Enhanced AutoML system, as well as ontology, suits their needs the most and create a classifier for their domain using that system. By choosing which Ontology-Enhanced AutoML system with which ontology to use to create a classifier, a user is better equipped to decide which factors are important for the specific classifier they would

like created, whether that is a high accuracy or a low wall-clock time.

4.4.4 Concerns

One of the largest sources of criticism we anticipate is that our class labels were produced specifically from the ontology provided. This is entirely true, though, we feel that this further emphasizes the capabilities of the system. Naturally, when enhancing an Ontology-Enhanced AutoML system with additional knowledge the given knowledge should pertain to the class labels. If the ontology does not provide insight into how to perform classification then there is no purpose in including the ontology. This would be akin to using an ontology describing astronomy to enhance a system that is being used to try to learn cooking recipes. Without an appropriate ontology, the additional knowledge gained by using ontologies is useless.

Over-fitting is also a concern some may have with our research. Naturally, any Ontology-Enhanced AutoML system that attains an accuracy of 1 is subject to the scrutiny of being over-fitted. However, examining the results of the Disqualifier/Qualifier examples that were removed we find that for the ABC dataset, approximately 13% of examples are filtered by the Disqualifier and are never learned from. For the Thingi10k dataset, approximately 17% of examples are filtered by the Disqualifier. The argument that the system is overfitting the dataset when such a large number of examples are not being learned from is weak.

Given that the task is classification, and the set of labels is binary, it makes sense that such high accuracy would occur. Rather, a large reason for such a high accuracy is caused by the guaranteed classification of examples that follow the rules of $Label_k$. Table 4.12. highlights the number of examples removed from the datasets for each system run.

An observation that we have been unable to explain is the varying amount of time Auto-Sklearn took to optimize runs 1, and 3 of the Thingi10k dataset, and runs 0, 1, 5, and 8 of the ABC dataset. These runs took close to ten times as long to optimize in both Auto-Sklearn and our Ontology-Enhanced AutoML system containing only the Feature Extraction mechanism. A clue we had at our disposal was the fact that the Ontology-

Dataset	Run	Training	Testing
ABC	0	1731	196
ABC	1	1724	203
ABC	2	1741	186
ABC	3	1736	191
ABC	4	1724	203
ABC	5	1733	1927
ABC	6	1743	1927
ABC	7	1745	1927
ABC	8	1736	191
ABC	9	1730	197
Thingi10k	0	1544	204
Thingi10k	1	1567	181
Thingi10k	2	1577	171
Thingi10k	3	1579	169
Thingi10k	4	1571	177
Thingi10k	5	1577	171
Thingi10k	6	1592	156
Thingi10k	7	1562	1748
Thingi10k	8	1574	174
Thingi10k	9	1589	159

Table 4.12: The number of examples that were disqualified for each run during experimentation.

Enhanced AutoML system, which contained the Disqualifier/Qualifier mechanism, did not see a similar increase in Optimization time. We ensured that no background processes were running during execution of these systems that might steal processor time from the experiment. We further examined the examples that were common amongst the outlying runs, and that had been removed by the Disqualifier. However, after examining all of the possible influences we were unable to conclude as to why this was the case.

Our current hypothesis is that there is some underlying pattern in these examples that erroneously causes Auto-Sklearn to initialize a hyperparameter configuration that is significantly slower than others during the earlier iterations of optimization. The act of introducing new features via our Feature Selection does not change the behaviour. This leads us to believe that it is something inherent in the other features of these examples. Further research may shed light on this discrepancy, however, we find that this specific issue is not so large as to detract from our results.

Overall we see these results as evidence that our proposed framework of ontology-enhanced AutoML has benefits compared to that of traditional AutoML. By enhancing an Ontology-Enhanced AutoML system with additional knowledge, more power over the learning process is given to a user, allowing them to incorporate their domain knowledge into a system. The increase in accuracy, as well as the improvement in optimization time, leads us to believe in the viability of our framework.

Chapter 5

Related Work

In this chapter we will look at already existing research and compare and contrast how these systems relate to our proposed concept of Ontology-Enhanced AutoML. In addition to AutoDL, which we discussed in Section 2.5, we also explore the fields of Semi-AutoML, Knowledge Graph Embeddings, Lifelong Learning, Pearl's Causal Calculus, and Ontology-Based ML.

5.1 AutoDL

AutoDL is the most easily comparable ML system to Ontology-Enhanced AutoML. AutoDL systems have been created to specifically utilize neural networks at runtime due to the expected format of allowed datasets [Liu+20a][Liu+20b][HZC21]. This is essentially an instance of Ontology-Enhanced AutoML where the transformation and incorporation components assume that there are certain properties present in an ontology.

Our definition of Ontology-Enhanced AutoML could be utilized to describe any AutoDL system as an Ontology-Enhanced AutoML system. Looking at any specific AutoDL system more closely, it is possible to define the transformation and incorporation components of an Ontology-Enhanced AutoML system to match the functionality of the AutoDL system. For example, $Trans$ could be implemented to pass a dataset through a pre-defined convolutional

neural network to extract features for each example.

Similarly, *Incorp* would allow for defining the modifications to the traditional AutoML pipeline which occur in AutoDL. The limiting of specific feature preprocessing methods is achieved by defining $inc_{pre_{data}}$ to modify the preprocessing methods of traditional AutoML. Defining inc_{meta} to utilize transfer learning in a metalearning component is another example of how AutoDL could be specified as an Ontology-Enhanced AutoML system. Thus, AutoDL is merely a specific instance of Ontology-Enhanced AutoML focused on the domain of sensor-type data.

5.2 Semi-AutoML

Semi-AutoML is a field that attempts to automate some components of an AutoML pipeline while requiring user input for other components [Shi+23][LTL21]. Alternatively called "Human-in-the-Loop" ML, Semi-AutoML works by allowing users to incorporate additional domain knowledge into a system manually at key points. The lower-level details of a Semi-AutoML system that are concerned with ML domain knowledge are automated since a user would not be expected to know these details [Mos+23][Zan19].

Common to all Semi-AutoML systems, a user plays a significant role during the learning process. The problem of how to integrate knowledge into an ML framework is a complex problem with a multitude of approaches [Den+20]. Wu et al. [Wu+22] identifies three possible roles of a human for incorporating additional knowledge into a system: supervisor, supervisee, and collaborator.

In what is called Active Learning [CGJ96], a user acts as an oracle that Semi-AutoML systems can ask questions for knowledge that a system is uncertain about. For example, a system may return two possible candidate class labels for classification and ask the user which is more correct. Systems like WhittleSearch [KPG12] show how human feedback can be used to tailor images generated by generative adversarial networks [Goo+20] to individual

tastes.

As a supervisee a human performs a given task, and the system learns from the human's behaviour. This is more popularly known as Imitation Learning [Hus+17]. In Imitation Learning, algorithms learn to mimic human behaviour by observing and imitating the actions of humans. In essence, a human acts as the input data for a learning system.

As a collaborator, a human works alongside a Semi-AutoML system to create a classifier. For example, a human may pick a desired Feature Preprocessing component recommended by a system or ignore the recommendation. Systems such as IBM's AutoAI Playground [Wan+20] work as an interface for users to select AutoML component decisions recommended by the system. In scenarios like this the system acts similarly to a recommendation system [Agg16].

Similar to Ontology-Enhanced AutoML, Semi-AutoML frameworks understand that domain knowledge is pivotal in learning accurately and efficiently. However, unlike Ontology-Enhanced AutoML, Semi-AutoML systems require constant human supervision to work properly. While both frameworks incorporate domain knowledge, only our Ontology-Enhanced AutoML framework can work entirely automated after inputs have been given to a system.

5.3 Knowledge Graph Embedding Frameworks

Knowledge Graph Embedding frameworks are frameworks that attempt to embed domain knowledge taxonomies into a low-dimensional representation for learning [Wan+17]. This is similar to our work in that both Knowledge Graph Embedding systems and Ontology-Enhanced AutoML frameworks attempt to utilize domain knowledge within an AutoML architecture. Both approaches attempt to take domain knowledge and leverage it when learning, specifically the relations of an ontology.

In essence, a knowledge graph embedding framework is typically used to create ML systems that are learned to represent knowledge from some domain. For example, Node2Vec

[GL16] is a system that learns vector embeddings from knowledge graphs. These embeddings are specifically represented as vectors of reals so that specific matrix operations can be used to operate on concepts contextually. Knowledge graph embeddings themselves are a learned representation of knowledge.

Both Knowledge Graph Embedding frameworks and Ontology-Enhanced AutoML frameworks aim to represent knowledge in a structured and semantically meaningful way. Knowledge graph embeddings capture the relationships between entities in a knowledge graph by embedding them into a continuous vector space, where entities and relations are represented as vectors. Similarly, Ontology-Enhanced AutoML utilizes ontologies, which are formal representations of knowledge that specify concepts and relationships between them.

Additionally, both approaches support inferencing and reasoning over knowledge. Knowledge graph embeddings enable reasoning by performing operations on entity and relation embeddings to infer new relationships or properties. Ontologies support inference by defining logical axioms and rules that can be used to derive new knowledge from existing knowledge.

However, a benefit of Ontology-Enhanced AutoML is that Ontology-Enhanced AutoML can create classifiers for a wide variety of similar domains while Knowledge Graph Embedding frameworks learn the semantic representations from the input data. Take for example TransE [Bor+13], and TransR [Lin+15]. These frameworks excel at capturing the relationships and structures within a knowledge graph, facilitating tasks like link prediction and entity classification. However, when confronted with data from domains that deviate significantly from the knowledge graph's structure, their performance might falter.

While these frameworks are powerful tools for knowledge graph tasks, their efficacy might diminish when confronted with data that diverges significantly from the structures represented in the graph. In such cases, Ontology-Enhanced AutoML techniques offer an alternative. By accepting domain-specific knowledge during the classifier creation process, Ontology-Enhanced AutoML enables the generation of classifiers tailored to the specifics of diverse domains, enhancing their adaptability and utility across a broader spectrum of

applications.

ML systems instantiated using knowledge graph embedding frameworks generate knowledge representations from specific ontological structures. Unlike Ontology-Enhanced AutoML, knowledge graph learning systems cannot be applied across domains when learning. The specific knowledge to be learned has to commonly include a measure of distance between concepts. For example, in Node2Vec, rather than learning automatically from any given knowledge graph, the concepts in a knowledge graph must first include a distance between relationships so that graph traversal algorithms can be applied.

5.4 Lifelong Learning

An interesting field that resembles Ontology-Enhanced AutoML is Lifelong Learning [Thr98] [Par+19]. Within the field of Lifelong Learning systems are designed to dynamically incorporate new knowledge while retaining previously acquired knowledge. One key aspect of Lifelong Learning is the notion of "plasticity," which refers to the ability of the classifier to adapt its structure and parameters in response to new information. Plasticity enables Lifelong Learning systems to continuously refine their representations and update their knowledge without catastrophic forgetting—where previously learned information is overwritten or lost. Another important concept in Lifelong Learning is transfer learning. Transfer learning helps reduce the required computational resources by enabling classifiers to reuse and repurpose knowledge learned from related tasks. Lifelong learning is similar to Ontology-Enhanced AutoML in that both frameworks attempt to utilize domain knowledge to generate classifiers across a variety of tasks. Within Ontology-Enhanced AutoML this knowledge is presented via an ontology whereas in Lifelong Learning the knowledge comes directly from several datasets.

The key difference between these two approaches is that Lifelong Learning continually updates a singular classifier over time while Ontology-Enhanced AutoML can produce clas-

sifiers for a wide variety of tasks on demand. When learning a new task the knowledge of a previous task within a Lifelong Learning system may be forgotten or disregarded over time. To relearn that knowledge previously learned datasets need to be re-inputted into the classifier to update the learned values.

Systems like ELLA [RE13] aim to overcome the issue of knowledge degradation by maintaining a corpus of previously learned tasks and using transfer learning to switch between classifiers as needed. A benefit provided by Ontology-Enhanced AutoML is that this corpus of classifiers does not need to be present to switch tasks. Within Ontology-Enhanced AutoML a user can simply switch which ontology they use to generate a classifier for a new task.

Lifelong Learning has also been attempted to be integrated into AutoML systems. Zheng et al. [Zhe+21] use a mechanism they call Lifelong Knowledge Anchors. Lifelong Knowledge Anchors act similar to a Knowledge Graph Embedding where all components of an AutoML pipeline are encoded as multidimensional vectors. These multidimensional vectors then form a continuous search space, as opposed to a discrete search space typically found in AutoML components. By implementing a Lifelong Evolutionary Algorithm wherein the embedded vectors can swap specific elements with one another Zheng et al. were able to out-compete other AutoDL frameworks in one round of the 2019 AutoDL competition.

5.5 Pearl's Do-Calculus

Here we cover the Do-Calculus proposed by Pearl [Pea94] as it pertains to our work. The Do-Calculus is a framework within the field of causal inference that provides a systematic approach for reasoning about cause-and-effect relationships. It is a powerful tool used in various disciplines, including statistics, ML [ZL23][Zha07], and biochemistry [Moh+22], to infer causal relationships from observational and experimental data. The calculus formalizes the process of identifying causal effects and making predictions based on those effects.

At first glance, the Do-Calculus appears to suit the needs of AutoML systems quite well. However, the Do-Calculus is more suited for determining confounders within a statistical sampling of real-world experiments. Confounders are variables in a study or experiment that are not measured, but that influence both the independent variables and the dependent variables. Because confounders are variables that are not measured the relationships between the dependent and independent variables may not accurately reflect reality. Consider, for example, an experiment to determine if excessive gambling affects the likelihood of heart attacks in individuals. When measured, the more a person gambles, the more likely they are to have a heart attack. A researcher may conclude that gambling increases the likelihood of heart attack, however, this may not be the case. Individuals with addictive tendencies such as gambling may be more likely to take up smoking since it is also an addictive behaviour. Whether a person smokes could be the actual influence on the likelihood of a heart attack, but the unmeasured relationship between smoking and gambling confounds things.

The Do-Calculus provides a framework for identifying the causality of a correlation based on the assumption of the existence of confounders within an experiment. Ontology-Enhanced AutoML, on the other hand, is able to express relationships which are far more complicated than statistical causality. Within a given dataset, for example, it may be possible to implement a mechanism for determining the causality of the features within a given dataset. However, a less complicated approach is already supported by our framework where a system could allow a domain expert to specify which concepts in a given ontology are assumed to be causal via relationships between concepts covering specific features. Alternatively, the engineer of an ontology could define some structure within $Additional_k$ to include the knowledge about which features are causal and which are not. One way this could be achieved is by specifying feature weights for specific types of classifiers, for example.

5.6 Ontology-Based ML

Ontology-based ML is an approach to ML that utilizes ontologies to enhance the effectiveness and precision of the learning process. It combines principles from information retrieval, knowledge representation, and semantic technologies to improve the accuracy of learned values by considering the meaning and relationships between concepts within a domain. This field focuses on learning relevant information from large datasets based on semantic understanding provided by ontologies.

A number of systems exist that use ontologies to learn from a dataset. For example, being able to predict weld-quality [Sve+20], improving communication between groups of agents [AFD06], natural language processing [CUM22], and classifying specific forms of epilepsy [Kas+14] are all systems that aim to use ontologies to perform ML for a given task.

This is similar to Ontology-Enhanced AutoML through its use of ontologies to generate additional knowledge about a dataset. Similar to how we defined, within $Additional_k$, the specific structures $Extract_k$ and $Label_k$, Ontology-based information retrieval focuses on extracting knowledge from a dataset using other structures that can also be defined within $Additional_k$. For example, Park et al.'s OLYBIA system [PJL07], uses semantic inference rules to automatically annotate images using a supplied ontology.

A key difference between Ontology-Based ML systems, and Ontology-Enhanced AutoML systems is the ability for Ontology-Enhanced AutoML systems to produce classifiers for a variety of tasks. In the case of OLYBIA, for example, while the supplied ontology can be modified to create different semantic inference rules the system as a whole is only able to be used to annotate images. Conversely, within Ontology-Enhanced AutoML, not only can the supplied ontology be altered, but the learning task can additionally be altered.

Within the ML spectrum, systems that fall within this category exist within the Knowledge-Enhanced ML range of frameworks. While these systems incorporate knowledge similar to Ontology-Enhanced AutoML, they are incapable of extending beyond a single task and are therefore not suitable for a broad range of application domains. Whenever a new task is

introduced, an entirely new system must be developed to meet the demands of the task. Compare this to Ontology-Enhanced AutoML, where a new classifier can be trained for a new task without having to do any additional work other than provide a new dataset and ontology.

Chapter 6

Conclusion and Future Work

In this chapter we present our conclusions out of the previous chapters. We then also present avenues for possible further research into the topic of Ontology-Enhanced AutoML.

6.1 Conclusion

In this thesis we presented a solution to the often poor performance of AutoML systems related to accuracy and computational time. We first conceptualized a spectrum for learning systems within ML. Using this spectrum, we theorized a possible approach to ML that allows for a near-arbitrary degree of generality to be decided during development. Using this approach, we created a generalized framework for incorporating ontologies into an AutoML system we call Ontology-Enhanced AutoML. We then created a first instance of our generalized framework focusing on two general methods for how to use an ontology and provided a specific ontology that could be used for datasets containing features pertaining to the field of computational geometry.

We then instantiated the two methods for incorporating domain knowledge into an Ontology-Enhanced AutoML system for our specific ontology. Using these mechanisms for ontological reasoning, we created an experiment where we tested two simple Ontology-Enhanced AutoML systems against Auto-Sklearn using two datasets containing features

pertaining to 3D modelling. From this experiment we found that our proposed generalized framework, for the given task, was able to outperform Auto-Sklearn in terms of accuracy, as well as optimization time.

Although we have seen that there was an increase in overall computation time using our approach, we provided some discussion regarding why this could be due to several factors that could be addressed.

6.2 Future Work

In this section we explore avenues to further refine and expand upon the Ontology-Enhanced AutoML framework presented in this thesis. Building upon the foundations laid out, there are several possible directions for advancement.

6.2.1 Using Additional Ontologies

An immediately apparent future goal for research in the field of Ontology-Enhanced AutoML is the use of additional ontologies. Within our experiments, we instantiated our proposed Ontology-Enhanced AutoML framework using an ontology focused on computational geometry. However, there are many fields that could benefit from the integration of ontologies into the AutoML process. By incorporating ontologies from diverse domains such as biology, or physics, researchers can tailor the framework to specific application domains, thereby enhancing its versatility and effectiveness. This approach would require adapting the framework to accommodate different ontology structures and semantics, or exploring techniques for integrating multiple ontologies seamlessly.

Naturally, to align with our vision, the generality of the researched ontologies would also play an effect in future research. Introducing additional ontologies that are too specific to apply to a variety of domains would create systems that are closer to PSML than AutoML, which may be undesirable.

Additionally, investigating methods for dynamically selecting or constructing ontologies based on the characteristics of the dataset or the requirements of the task could further optimize the AutoML process. This could work similarly to that of Knowledge Graph Embedding systems, or Lifelong Learning systems where a supplied ontology could be adapted or extended based on a received dataset.

6.2.2 Enhanced Ontological Reasoning

As discussed previously, one of the most prominent limitations of our current system is that the Feature Extraction is hard-coded into the system. Consequently, this specific system can currently handle only datasets that specifically contain 3D models in a b-reps format. This is closer to being PSML than would be ideal.

One strategy to enhance a system's generality involves enabling it to interpret a wide range of ontologies, thereby accommodating diverse domains. By incorporating support for OWL syntax or similar ontology markup languages, the system gains the capability to understand and utilize ontological representations of domain knowledge. Tools like Protégé [Mus15] provide a mechanism for this integration, offering a platform for importing and utilizing arbitrary ontologies, thus establishing a bridge between domain-specific knowledge and machine learning processes. Naturally, one of the more challenging aspects is establishing a mapping from ontology definitions to specific functionality, such as how to interpret which features to extract, and how they specifically can be extracted.

There are also additional methods for getting more use out of an ontology provided to a system. By implementing functionality allowing interpretation of arbitrary ontologies, further research can be done to study how that knowledge can be used. Generalizing to previous research in AutoML we can see how Ontology-Enhanced AutoML could be implemented to enhance the capabilities of each stage of the traditional AutoML framework. Specifically, we think that defining structures expected to be present within $Additional_k$, similar to how $Extract_k$ and $Label_k$ were defined, could lead to further increases in accuracy, or decreases

in computation time of an Ontology-Enhanced AutoML system.

For use in Metalearning, $Additional_k$ could be expected to contain a variety of internal structures. For example, knowledge indicating which learning algorithms are more likely to be successful can be represented in various forms, such as specifying a specific similarity measure to use when comparing meta-features for algorithm selection as described by Feurer et al. [Feu+22].

$Additional_k$ could also be expected to contain specific structures as a way to provide an Ontology-Enhanced AutoML system with input weights for data preprocessing. If a user is aware of how relevant certain examples are to a problem there is no need for an Ontology-Enhanced AutoML system to relearn that relevance. Rather than having an Ontology-Enhanced AutoML system decide what input weights should be based on class imbalances of the predictor feature, that knowledge could be made explicit via an ontology. This would allow an Ontology-Enhanced AutoML system to make a more informed decision during optimization.

Furthermore, $Additional_k$ could be expected to contain structures that can be used to modify postprocessing through the use of techniques such as error analysis. This could be achieved by defining structures that contain insights into the semantics and context of specific classifiers. By encoding domain-specific knowledge about acceptable prediction ranges, decision thresholds, and domain-specific evaluation metrics within an ontology, Ontology-Enhanced AutoML systems could ensure that calibrated classifiers align with domain requirements and exhibit desirable behaviour in real-world applications.

One way we foresee the field of Ontology-Enhanced AutoML progressing is via a variety of domain-specific systems. What we mean by this is that specific Ontology-Enhanced AutoML systems will be built for large and similar domains that can utilize the same ontology across various fields. For example, our specific instantiation of our Ontology-Enhanced AutoML framework uses an ontology pertaining to computational geometry. By incorporating an ontology relating to computational geometry any domain that pertains to real-world ge-

ometry can benefit from the high-level knowledge stored within our ontology. Construction, 3D modelling, architecture, etc. are all separate fields that rely on computational geometry to function. Our first instance would be properly equipped to produce a predictive classifier for any of these fields because knowledge about 3D objects is universal across these fields.

If we relate our idea of a domain-specific system back to our discussion of an ML spectrum we see that this is a viable way for a user to pick and choose the degree of knowledge incorporated within a system suited to their needs. If a user of an Ontology-Enhanced AutoML system requires less domain knowledge to be included, they would be able to choose an Ontology-Enhanced AutoML system that expects specific structures to be defined within $Additional_k$ that relate to their field of interest. Developers could then, in turn, detail the specific requirements of their developed Ontology-Enhanced AutoML systems. Ontology-Enhanced AutoML systems with more stringent requirements could document more specialized structures expected to be present within $Additional_k$ so that users can make an informed decision about whether they should use a particular Ontology-Enhanced AutoML system or not.

Bibliography

[ADF85] Silvia Ansaldi, Leila De Floriani, and Bianca Falcidieno. "Geometric Modeling of Solid Objects by Using a Face Adjacency Graph Representation". In: *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '85. New York, NY, USA: Association for Computing Machinery, 1985, pp. 131–139.

[AFD06] Mohsen Afsharchi, Behrouz H Far, and Jörg Denzinger. "Ontology-Guided Learning to Improve Communication Between Groups of Agents". In: *Proceedings of the fifth international joint conference on Autonomous agents and multiagent systems*. 2006, pp. 923–930.

[Agg16] Charu C Aggarwal. *Recommender systems*. Vol. 1. Springer, 2016.

[AH09] Grigoris Antoniou and Frank van Harmelen. "Web Ontology Language: OWL". In: *Handbook on Ontologies*. Ed. by Steffen Staab and Rudi Studer. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 91–110.

[Aha92] David W Aha. "Generalizing From Case Studies: A Case Study". In: *Machine Learning Proceedings 1992*. Elsevier, 1992, pp. 1–10.

[AI23] Open AI. *GPT-4 Technical Report*. Tech. rep. Accessed: 2024-02-13. OpenAI (2023), 2023.

[All14] David H Allen. *How Mechanics Shaped The Modern World*. Springer, 2014, pp. 176–177.

[AMD22] Debra J Audus, Austin McDannald, and Brian DeCost. “Leveraging Theory for Enhanced Machine Learning”. In: *ACS Macro Letters* 11.9 (2022), pp. 1117–1122.

[Baa03] Franz Baader. *The description logic handbook: Theory, implementation and applications*. Cambridge university press, 2003.

[Bau75] Bruce G. Baumgart. “A Polyhedron Representation for Computer Vision”. In: *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition*. AFIPS ’75. Anaheim, California: Association for Computing Machinery, 1975, pp. 589–596.

[BB13] Mauro Brunato and Roberto Battiti. “Learning and intelligent optimization (LION): one ring to rule them all”. In: *Proc. VLDB Endow.* 6.11 (Aug. 2013), pp. 1176–1177.

[Ber+11] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. “Algorithms for hyper-parameter optimization”. In: *Proceedings of the 24th International Conference on Neural Information Processing Systems*. NIPS’11. Red Hook, NY, USA: Curran Associates Inc., 2011, pp. 2546–2554.

[Bha+20] Shreyansh Bhatt, Amit Sheth, Valerie Shalin, and Jinjin Zhao. “Knowledge Graph Semantic Enhancement of Input Data for Improving AI”. In: *IEEE Internet Computing* 24.2 (2020), pp. 66–72.

[Bla98] Paul Black. *Dictionary of Algorithms and Data Structures*. `http://www.nist.gov/dads`. Accessed: 2024-02-13. 1998.

[BM12] M. Lo Brutto and P. Meli. “Computer Vision Tools for 3D Modelling in Archaeology”. In: *International Journal of Heritage in the Digital Era* 1.1_suppl (2012), pp. 1–6.

[Bor+13] Antoine Bordes, Nicolas Usunier, Alberto Garcia-Durán, Jason Weston, and Oksana Yakhnenko. "Translating embeddings for modeling multi-relational data". In: *Proceedings of the 26th International Conference on Neural Information Processing Systems - Volume 2*. NIPS'13. Red Hook, NY, USA: Curran Associates Inc., 2013, pp. 2787–2795.

[Bor96] Alex Borgida. "On the relative expressiveness of description logics and predicate logics". In: *Artificial Intelligence* 82.1 (1996), pp. 353–367.

[Bot+02] Mario Botsch, Stefan Steinberg, Stefan Bischoff, and Leif Kobbelt. *OpenMesh: A Generic and Efficient Polygon Mesh Data Structure*. `https://www.graphics.rwth-aachen.de/media/papers/openmesh1.pdf`. Accessed: 2024-02-13. 2002.

[Bra+08] Pavel Brazdil, Christophe Giraud Carrier, Carlos Soares, and Ricardo Vilalta. *Metalearning: Applications to Data Mining*. Springer Science & Business Media, 2008.

[BSI19] A. Baldominos, Y. Saez, and P. Isasi. "A Survey of Handwritten Character Recognition with MNIST and EMNIST". In: *Applied Sciences* 9.15 (2019), p. 3169.

[Büc+17] Thore M. Bücking, Emma R. Hill, James L. Robertson, Efthymios Maneas, Andrew A. Plumb, and Daniil I. Nikitichev. "From medical imaging data to 3D printed anatomical models". In: *PLOS ONE* 12.5 (May 2017), pp. 1–10.

[BVR23] Rafael Barbudo, Sebastián Ventura, and José Raúl Romero. "Eight Years of AutoML: Categorisation, Review and Trends". In: *Knowledge and Information Systems* 65.12 (2023), pp. 5097–5149.

[Cao+21] Xianshuai Cao, Yuliang Shi, Han Yu, Jihu Wang, Xinjun Wang, Zhongmin Yan, and Zhiyong Chen. "DEKR: description enhanced knowledge graph for machine learning method recommendation". In: *Proceedings of the 44th Interna-*

tional ACM SIGIR Conference on Research and Development in Information Retrieval. 2021, pp. 203–212.

[Car+04] Rich Caruana, Alexandru Niculescu-Mizil, Geoff Crew, and Alex Ksikes. "Ensemble Selection from Libraries of Models". In: *Proceedings of the twenty-first international conference on Machine learning*. 2004, pp. 1–18.

[CGJ96] David A. Cohn, Zoubin Ghahramani, and Michael I. Jordan. "Active learning with statistical models". In: *J. Artif. Int. Res.* 4.1 (Mar. 1996), pp. 129–145.

[Cha20] Xuqing Chai. "Diagnosis Method of Thyroid Disease Combining Knowledge Graph and Deep Learning". In: *IEEE Access* 8 (2020), pp. 149787–149795.

[CM21] Fabio Gagliardi Cozman and Hugo Neri Munhoz. "Some Thoughts on Knowledge-Enhanced Machine Learning". In: *International Journal of Approximate Reasoning* 136 (2021), pp. 308–324.

[Cod70] Edgar F Codd. "A Relational Model of Data for Large Shared Data Banks". In: *Communications of the ACM* 13.6 (1970), pp. 377–387.

[Com18] Blender Online Community. *Blender - A 3D Modelling and Rendering Package*. `http://www.blender.org`. Accessed: 2024-02-13. Blender Foundation, 2018.

[CUM22] Philipp Cimiano, Christina Unger, and John McCrae. *Ontology-based interpretation of natural language*. Springer Nature, 2022.

[CW19] Christopher M Childs and Newell R Washburn. "Embedding Domain Knowledge for Machine Learning of Complex Material Systems". In: *MRS Communications* 9.3 (2019), pp. 806–820.

[Den+20] Changyu Deng, Xunbi Ji, Colton Rainey, Jianyu Zhang, and Wei Lu. "Integrating Machine Learning with Human Knowledge". In: *iScience* 23.11 (2020). online only `https://doi.org/10.1016/j.isci.2020.101656`.

[DS19] Alessandro Daniele and Luciano Serafini. “Knowledge Enhanced Neural Networks”. In: *PRICAI 2019: Trends in Artificial Intelligence*. Ed. by Abhaya C. Nayak and Alok Sharma. Springer International Publishing, 2019, pp. 542–554.

[EA12] Philippe Esling and Carlos Agon. “Time-series data mining”. In: *ACM Comput. Surv.* 45.1 (Dec. 2012), pp. 1–34.

[FD03] William A Figueiredo and Air Force Research Labs Edwards AFB CA Propulsion Directorate. “High Thrust to Weight Bi-propellant Re-entry Vehicle Thrust Vector Control thru Micro-Miniaturization”. In: (2003). online only `https://doi.org/10.2514/6.2003-525`.

[Fen+00] Dieter Fensel, Ian Horrocks, Frank Van Harmelen, Stefan Decker, Michael Erdmann, and Michel Klein. “OIL in a Nutshell”. In: *Knowledge Engineering and Knowledge Management Methods, Models, and Tools*. Ed. by Rose Dieng and Olivier Corby. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 1–16.

[Feu+15] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. “Efficient and Robust Automated Machine Learning”. In: *Advances in Neural Information Processing Systems*. 2015, pp. 2962–2970.

[Feu+22] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. “Auto-sklearn 2.0: hands-free AutoML via meta-learning”. In: *J. Mach. Learn. Res.* 23.1 (Jan. 2022), pp. 11936–11996.

[FKH18] Stefan Falkner, Aaron Klein, and Frank Hutter. “BOHB: Robust and Efficient Hyperparameter Optimization at Scale”. In: *Proceedings of the 35th International Conference on Machine Learning*. 2018, pp. 1436–1445.

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[GH95] Patrick J Grother and KK Hanaoka. "NIST special database 19". In: *Handprinted forms and characters database, National Institute of Standards and Technology* 10 (1995), pp. 1–69.

[Ghi+20] Alina Ghilan, Aurica P. Chiriac, Loredana E. Nita, Alina G. Rusu, Iordana Neamtu, and Vlad Mihai Chiriac. "Trends in 3D Printing Processes for Biomedical Field: Opportunities and Challenges". In: *Journal of Polymers and the Environment* 28.5 (May 2020), pp. 1345–1367. ISSN: 1572-8919.

[GL16] Aditya Grover and Jure Leskovec. "node2vec: Scalable feature learning for networks". In: *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. 2016, pp. 855–864.

[Gla89] Andrew S Glassner. *An Introduction To Ray Tracing*. Morgan Kaufmann, 1989.

[Gom+09] Abel J. P. Gomes, Irina Voiculescu, Joaquim Jorge, Brian Wyvill, and Callum Galbraith. *Continuation Methods*. Ed. by Abel J. P. Gomes, Irina Voiculescu, Joaquim Jorge, Brian Wyvill, and Callum Galbraith. London: Springer London, 2009, pp. 145–186.

[Goo+20] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. "Generative adversarial networks". In: *Commun. ACM* 63.11 (Oct. 2020), pp. 139–144.

[Gro16] Michael Groenendyk. "Cataloging the 3D web: the availability of educational 3D models on the internet". In: *Library Hi Tech* 34.2 (2016), pp. 239–258. ISSN: 0737-8831.

[Guy+19] Isabelle Guyon, Lisheng Sun-Hosoya, Marc Boullé, Hugo Jair Escalante, Sergio Escalera, Zhengying Liu, Damir Jajetic, Bisakha Ray, Mehreen Saeed, Michèle Sebag, Alexander Statnikov, Wei-Wei Tu, and Evelyne Viegas. "Analysis of the AutoML Challenge Series 2015–2018". In: *Automated Machine Learning: Meth-*

ods, Systems, Challenges. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Cham: Springer International Publishing, 2019, pp. 177–219.

[Hal+09] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. "The WEKA Data Mining Software: An Update". In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[Her23] Óscar Hernández-Muñoz. "Analysis of Digitized 3D Models Published by Archaeological Museums". In: *Heritage* 6.5 (2023), pp. 3885–3902. ISSN: 2571-9408.

[HHL11] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. "Sequential Model-Based Optimization for General Algorithm Configuration". In: *Proceedings of the 5th International Conference on Learning and Intelligent Optimization*. LION'05. Rome, Italy: Springer-Verlag, 2011, pp. 507–523.

[HKV19] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, eds. *Automated Machine Learning: Methods, Systems, Challenges*. Foreword by Zoubin Ghahramani. Springer International Publishing, 2019.

[HL23] Xi Hu and Kaijian Liu. "Structural Deterioration Knowledge Ontology towards Physics-Informed Machine Learning for Enhanced Bridge Deterioration Prediction". In: *Journal of Computing in Civil Engineering* 37.1 (2023). online only `https://doi.org/10.1061/(ASCE)CP.1943-5487.0001066`.

[Hus+17] Ahmed Hussein, Mohamed Medhat Gaber, Eyad Elyan, and Chrisina Jayne. "Imitation Learning: A Survey of Learning Methods". In: *ACM Comput. Surv.* 50.2 (Apr. 2017), pp. 1–35.

[HZC21] Xin He, Kaiyong Zhao, and Xiaowen Chu. "AutoML: A Survey of the State-of-the-Art". In: *Knowledge-Based Systems* 212 (2021). online only `https://doi.org/10.1016/j.knosys.2020.106622`.

[Jam+13] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An Introduction to Statistical Learning*. Vol. 112. Springer, 2013.

[Jum+21] John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, and Anna Potapenko. "Highly accurate protein structure prediction with AlphaFold". In: *Nature* 596.7873 (2021), pp. 583–589.

[JV13] Salim Jouili and Valentin Vansteenberghe. "An Empirical Comparison of Graph Databases". In: *2013 International Conference on Social Computing*. 2013, pp. 708–715.

[Kar+17] Anuj Karpatne, Gowtham Atluri, James H Faghmous, Michael Steinbach, Arindam Banerjee, Auroop Ganguly, Shashi Shekhar, Nagiza Samatova, and Vipin Kumar. "Theory-Guided Data Science: A New Paradigm for Scientific Discovery From Data". In: *IEEE Transactions on knowledge and data engineering* 29.10 (2017), pp. 2318–2331.

[Kas+14] Yohannes Kassahun, Roberta Perrone, Elena De Momi, Elmar Berghöfer, Laura Tassi, Maria Paola Canevini, Roberto Spreafico, Giancarlo Ferrigno, and Frank Kirchner. "Automatic Classification of Epilepsy Types using Ontology-Based and Genetics-Based Machine Learning". In: *Artificial Intelligence in Medicine* 61.2 (2014), pp. 79–88.

[Ken83] John T. Kent. "Information Gain and A General Measure of Correlation". In: *Biometrika* 70.1 (Apr. 1983), pp. 163–173.

[Kir08] "Level of MeasurementLevel of measurement". In: *Encyclopedia of Public Health*. Ed. by Wilhelm Kirch. Dordrecht: Springer Netherlands, 2008, pp. 851–852. ISBN: 978-1-4020-5614-7. DOI: 10.1007/978-1-4020-5614-7_1971. URL: https://doi.org/10.1007/978-1-4020-5614-7_1971.

[KJC16] Jungtaek Kim, Jongheon Jeong, and Seungjin Choi. "AutoML Challenge: AutoML Framework Using Random Space Partitioning Optimizer". In: *Workshop on Automated Machine Learning (AutoML), ICML*. 2016, pp. 1–4.

[KKK22] Dohyung Kim, Jahwan Koo, and Ung-Mo Kim. “A Survey on Automated Machine Learning: Problems, Methods and Frameworks”. In: *Human-Computer Interaction. Theoretical Approaches and Design Methods*. Ed. by Masaaki Kurosu. Cham: Springer International Publishing, 2022, pp. 57–70.

[KLW95] Michael Kifer, Georg Lausen, and James Wu. “Logical Foundations of Object-Oriented and Frame-Based Languages”. In: *J. ACM* 42.4 (July 1995), pp. 741–843.

[Koc+19] Sebastian Koch, Albert Matveev, Zhongshi Jiang, Francis Williams, Alexey Artemov, Evgeny Burnaev, Marc Alexa, Denis Zorin, and Daniele Panozzo. “ABC: A Big CAD Model Dataset for Geometric Deep Learning”. In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 9601–9611.

[KPG12] Adriana Kovashka, Devi Parikh, and Kristen Grauman. “WhittleSearch: Image search with relative attribute feedback”. In: *2012 IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 2973–2980.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Advances in Neural Information Processing Systems*. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012, pp. 84–90.

[KT11] Hian Chye Koh and Gerald Tan. “Data Mining Applications in Healthcare”. In: *Journal of healthcare information management* 19.2 (2011), pp. 64–72.

[LG07] Jun won Lee and Christophe Giraud-Carrier. “Transfer Learning in Decision Trees”. In: *2007 International Joint Conference on Neural Networks*. 2007, pp. 726–731.

[LH18] Marius Lindauer and Frank Hutter. "Warmstarting of Model-Based Algorithm Configuration". In: *Thirty-Second AAAI Conference on Artificial Intelligence*. 2018, pp. 1355–1362.

[Li+18] Lisha Li, Kevin Jamieson, Giulia DeSalvo, Afshin Rostamizadeh, and Ameet Talwalkar. "Hyperband: A Novel Bandit-Based Approach to Hyperparameter Optimization". In: *Journal of Machine Learning Research* 18.185 (2018), pp. 1–52.

[Lik32] R. Likert. "A Technique for the Measurement of Attitudes". In: *Archives of Psychology* 22.140 (1932), p. 55.

[Lin+15] Yankai Lin, Zhiyuan Liu, Maosong Sun, Yang Liu, and Xuan Zhu. "Learning Entity and Relation Embeddings for Knowledge Graph Completion". In: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 29. 2015, pp. 2181–2187.

[Liu+20a] Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Isabelle Guyon, JCJ Junior, Meysam Madadi, and Sebastien Treguer. *How Far Are We From True AutoML: Reflection From Winning Solutions and Results of AutoDL Challenge*. `https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_51.pdf`. Accessed: 2024-05-28. 2020.

[Liu+20b] Zhengying Liu, Zhen Xu, Shangeth Rajaa, Meysam Madadi, Julio C. S. Jacques Junior, Sergio Escalera, Adrien Pavao, Sebastien Treguer, Wei-Wei Tu, and Isabelle Guyon. "Towards Automated Deep Learning: Analysis of the AutoDL challenge series 2019". In: *Proceedings of the NeurIPS 2019 Competition and Demonstration Track*. Ed. by Hugo Jair Escalante and Raia Hadsell. Vol. 123. Proceedings of Machine Learning Research. PMLR, Dec. 2020, pp. 242–252.

[Liu+21] Zhengying Liu, Adrien Pavao, Zhen Xu, Sergio Escalera, Fabio Ferreira, Isabelle Guyon, Sirui Hong, Frank Hutter, Rongrong Ji, Julio C. S. Jacques Ju-

nior, Ge Li, Marius Lindauer, Zhipeng Luo, Meysam Madadi, Thomas Nierhoff, Kangning Niu, Chunguang Pan, Danny Stoll, Sebastien Treguer, Jin Wang, Peng Wang, Chenglin Wu, Youcheng Xiong, Arbër Zela, and Yang Zhang. "Winning Solutions and Post-Challenge Analyses of the ChaLearn AutoDL Challenge 2019". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9 (2021), pp. 3108–3125.

[LTL21] Yue Liu, Wenjie Tian, and Shuang Li. "Meta-data Augmentation Based Search Strategy Through Generative Adversarial Network for AutoML Model Selection". In: *Advances in Knowledge Discovery and Data Mining*. Ed. by Kamal Karlapalem, Hong Cheng, Naren Ramakrishnan, R. K. Agrawal, P. Krishna Reddy, Jaideep Srivastava, and Tanmoy Chakraborty. Cham: Springer International Publishing, 2021, pp. 312–324.

[Mac91] Robert M. MacGregor. "Inside the LOOM Description Classifier". In: *SIGART Bull.* 2.3 (June 1991), pp. 88–92.

[MB08] Kenton McHenry and Peter Bajcsy. *An Overview of 3D Data Content, File Formats and Viewers*. Tech. rep. National Center for Supercomputing Applications, 2008.

[Moh+22] Sara Mohammad-Taheri, Jeremy Zucker, Charles Tapley Hoyt, Karen Sachs, Vartika Tewari, Robert Ness, and Olga Vitek. "Do-Calculus Enables Estimation of Causal Effects in Partially Observed Biomolecular Pathways". In: *Bioinformatics* 38 (June 2022), pp. 350–358.

[Moo+19] Bethany M. Moore, Peipei Wang, Pengxiang Fan, Bryan Leong, Craig A. Schenck, John P. Lloyd, Melissa D. Lehti-Shiu, Robert L. Last, Eran Pichersky, and Shin-Han Shiu. "Robust predictions of specialized metabolism genes through machine learning". In: *Proceedings of the National Academy of Sciences* 116.6 (2019), pp. 2344–2353.

[Mos+23] Eduardo Mosqueira-Rey, Elena Hernández-Pereira, David Alonso-Ríos, José Bobes-Bascarán, and Ángel Fernández-Leal. "Human-in-the-loop machine learning: a state of the art". In: *Artificial Intelligence Review* 56.4 (2023), pp. 3005–3054.

[Mot05] Boris Motik. "On the Properties of Metamodeling in OWL". In: *The Semantic Web – ISWC 2005*. Ed. by Yolanda Gil, Enrico Motta, V. Richard Benjamins, and Mark A. Musen. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 548–562.

[MRT18] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. 2nd. MIT Press, 2018.

[Mue20] Jody Muelaner. *Implicit Modelling For Complex Geometry*. `https://www.3dcadworld.com/implicit-modelling-for-complex-geometry/`. Accessed: 2024-02-13. 2020.

[Mus15] Mark A. Musen. "The Protégé Project: A Look Back and A Look Forward". In: *AI Matters* 1.4 (2015), pp. 4–12.

[NLA24] Felix Neutatz, Marius Lindauer, and Ziawasch Abedjan. "AutoML in Heavily Constrained Applications". In: *The VLDB Journal* 33.4 (July 2024), pp. 957–979.

[NP19] Thiloshon Nagarajah and Guhanathan Poravi. "A Review on Automated Machine Learning (AutoML) Systems". In: *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. 2019, pp. 1–6.

[Pac99] János Pach. "Geometric Graph Theory". In: *Surveys in Combinatorics, 1999*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1999, pp. 167–200.

[Par+19] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. “Continual lifelong learning with neural networks: A review”. In: *Neural networks* 113 (2019), pp. 54–71.

[Pea94] Judea Pearl. “A Probabilistic Calculus of Actions”. In: *Uncertainty in Artificial Intelligence*. Ed. by Ramon Lopez de Mantaras and David Poole. San Francisco (CA): Morgan Kaufmann, 1994, pp. 454–462.

[Ped+11] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. “Scikit-learn: Machine Learning in Python”. In: *Journal of Machine Learning Research* 12.85 (2011), pp. 2825–2830.

[Per+15] Alexander Perzylo, Nikhil Somani, Markus Rickert, and Alois Knoll. “An ontology for CAD data and geometric constraints as a link between product models and semantic robot task descriptions”. In: *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 4197–4203.

[PJL07] Kyung-Wook Park, Jin-Woo Jeong, and Dong-Ho Lee. “OLYBIA: Ontology-Based Automatic Image Annotation System Using Semantic Inference Rules”. In: *Advances in Databases: Concepts, Systems and Applications*. Ed. by Ramamohanarao Kotagiri, P. Radha Krishna, Mukesh Mohania, and Ekawit Nantajeewarawat. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 485–496.

[Pła+16] Paweł Pławiak, Tomasz Sośnicki, Michał Niedźwiecki, Zbisław Tabor, and Krzysztof Rzecki. “Hand Body Language Gesture Recognition Based on Signals From Specialized Glove and Machine Learning Algorithms”. In: *IEEE Transactions on Industrial Informatics* 12.3 (2016), pp. 1104–1113.

[Pol12] Robi Polikar. “Ensemble Learning”. In: *Ensemble Machine Learning: Methods and Applications*. New York, NY: Springer New York, 2012, pp. 1–34.

[PY16] Bernd S. Palm and Alf Yarwood. *Introduction to AutoCAD 2017: 2D and 3D Design*. USA: Routledge, 2016.

[RC11] Radu Bogdan Rusu and Steve Cousins. “3D Is Here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1–4.

[RE13] Paul Ruvolo and Eric Eaton. “ELLA: An Efficient Lifelong Learning Algorithm”. In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 1. Atlanta, Georgia, USA: PMLR, June 2013, pp. 507–515.

[Rem03] Fabio Remondino. “From point cloud to surface: the modeling and visualization problem”. en. In: *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*. Ed. by Armin Grün, S. Murai, J. Niederoest, and Fabio Remondino. Vol. XXXIV-5/W10. Tarasp-Vulpera: ISPRS, 2003, pp. 1–11.

[Ric76] John R Rice. “The Algorithm Selection Problem”. In: *Advances in computers*. Vol. 15. Elsevier, 1976, pp. 65–118.

[Roe+19] Rebecca Roelofs, Vaishaal Shankar, Benjamin Recht, Sara Fridovich-Keil, Moritz Hardt, John Miller, and Ludwig Schmidt. “A Meta-Analysis of Overfitting in Machine Learning”. In: *Advances in Neural Information Processing Systems*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Vol. 32. Curran Associates, Inc., 2019, pp. 9179–9189.

[Ros06] Riccardo Rosati. “The Limits of Querying Ontologies”. In: *Database Theory – ICDT 2007*. Ed. by Thomas Schwentick and Dan Suciu. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 164–178.

[Rus+15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. "Imagenet Large Scale Visual Recognition Challenge". In: *International journal of computer vision* 115.3 (2015), pp. 211–252.

[Sar+22] Avadhut Sardeshmukh, Sreedhar Reddy, B P. Gautham, and Ankit Agrawal. "Machine Learning for Materials Science (MLMS)". In: *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 4902–4903.

[Sch+22] Teseo Schneider, Yixin Hu, Xifeng Gao, Jérémie Dumas, Denis Zorin, and Daniele Panozzo. "A Large-Scale Comparison of Tetrahedral and Hexahedral Elements for Solving Elliptic PDEs with the Finite Element Method". In: *ACM Trans. Graph.* 41.3 (Mar. 2022), pp. 1–14.

[Shi+23] Yuqi Shi, Jiaxi Lin, Jinzhou Zhu, Jingwen Gao, Lu Liu, Minyue Yin, Chenyan Yu, Xiaolin Liu, Yu Wang, and Chunfang Xu. "Predicting the Recurrence of Common Bile Duct Stones After ERCP Treatment with Automated Machine Learning Algorithms". In: *Digestive Diseases and Sciences* 68.7 (2023), pp. 2866–2877.

[Sil+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play". In: *Science* 362.6419 (2018), pp. 1140–1144.

[SLA12] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. "Practical Bayesian Optimization of Machine Learning Algorithms". In: *Advances in Neural Information*

Processing Systems. Ed. by F. Pereira, C.J. Burges, L. Bottou, and K.Q. Weinberger. Vol. 25. Curran Associates, Inc., 2012, pp. 2951–2959.

[Sou14] Cézar Roberto Souza. *The Accord .NET Framework*. `https://github.com/accord-net/framework`. Accessed: 2024-05-28. 2014.

[Str06] Ian Stroud. *Boundary Representation Modelling Techniques*. Springer Science & Business Media, 2006.

[Stu02] Gerd Stumme. "Using Ontologies and Formal Concept Analysis for Organizing Business Knowledge". In: *Wissensmanagement mit Referenzmodellen*. Springer, 2002, pp. 163–174.

[Sve+20] Yulia Svetashova, Baifan Zhou, Tim Pychynski, Stefan Schmidt, York Sure-Vetter, Ralf Mikut, and Evgeny Kharlamov. "Ontology-Enhanced Machine Learning: A Bosch Use Case of Welding Quality Monitoring". In: *The Semantic Web–ISWC 2020: 19th International Semantic Web Conference, Athens, Greece, November 2–6, 2020, Proceedings, Part II 19*. Springer. 2020, pp. 531–550.

[Tho+13] Chris Thornton, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. "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*. 2013, pp. 847–855.

[Thr98] Sebastian Thrun. "Lifelong learning algorithms". In: *Learning to learn*. Springer, 1998, pp. 181–209.

[TS10] Lisa Torrey and Jude Shavlik. "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[TTS06] Oren Tropp, Ayellet Tal, and Ilan Shimshoni. "A Fast Triangle-Triangle Intersection Test for Collision Detection". In: *Computer Animation and Virtual Worlds* 17.5 (2006), pp. 527–535.

[Van19] Joaquin Vanschoren. “Meta-learning”. In: *Automated Machine Learning: Methods, Systems, Challenges*. Springer, Cham, 2019, pp. 35–61.

[Von+21] Laura Von Rueden, Sebastian Mayer, Katharina Beckh, Bogdan Georgiev, Sven Giesselbach, Raoul Heese, Birgit Kirsch, Julius Pfrommer, Annika Pick, and Rajkumar Ramamurthy. “Informed Machine Learning–A Taxonomy and Survey of Integrating Prior Knowledge into Learning Systems”. In: *IEEE Transactions on Knowledge and Data Engineering* 35.1 (2021), pp. 614–633.

[Wal+09] Ingo Wald, William R. Mark, Johannes Günther, Solomon Boulos, Thiago Ize, Warren Hunt, Steven G. Parker, and Peter Shirley. “State of the Art in Ray Tracing Animated Scenes”. In: *Computer Graphics Forum* 28.6 (2009), pp. 1691–1722.

[Wan+17] Quan Wang, Zhendong Mao, Bin Wang, and Li Guo. “Knowledge Graph Embedding: A Survey of Approaches and Applications”. In: *IEEE Transactions on Knowledge and Data Engineering* 29.12 (2017), pp. 2724–2743.

[Wan+20] Dakuo Wang, Parikshit Ram, Daniel Karl I. Weidele, Sijia Liu, Michael Muller, Justin D. Weisz, Abel Valente, Arunima Chaudhary, Dustin Torres, Horst Samulowitz, and Lisa Amini. “AutoAI: Automating the End-to-End AI Lifecycle with Humans-in-the-Loop”. In: *Companion Proceedings of the 25th International Conference on Intelligent User Interfaces*. New York, NY, USA: Association for Computing Machinery, 2020, pp. 77–78.

[WEG87] Svante Wold, Kim Esbensen, and Paul Geladi. “Principal Component Analysis”. In: *Chemometrics and Intelligent Laboratory Systems* 2.1 (1987). Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists, pp. 37–52.

[Wei] Eric W Weisstein. *Heron’s Formula*. `https://mathworld.wolfram.com/HeronsFormula.html`. Accessed: 2024-02-13.

[Wer+23] Edgar Werthen, Daniel Ribnitzky, David Zerbst, Martin Kühn, and Christian Hühne. “Aero-Structural Coupled Optimization of a Rotor Blade for an Up-scaled 25 MW Reference Wind Turbine”. In: *Journal of Physics: Conference Series*. Vol. 2626. online only `https://doi.org/10.1088/1742-6596/2626/1/012012`. IOP Publishing. 2023.

[WM95] David H Wolpert and William G Macready. *No Free Lunch Theorems for Search*. Tech. rep. Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.

[WM97] David H Wolpert and William G Macready. “No Free Lunch Theorems for Optimization”. In: *IEEE transactions on evolutionary computation* 1.1 (1997), pp. 67–82.

[Wol92] David H. Wolpert. “Stacked Generalization”. In: *Neural Networks* 5.2 (1992), pp. 241–259.

[WR16] Nicholas Wagner and James M Rondinelli. “Theory-Guided Machine Learning in Materials Science”. In: *Frontiers in Materials* 3 (2016), pp. 1–28.

[WS92] William A. Woods and James G. Schmolze. “The KL-ONE Family”. In: *Computers & Mathematics with Applications* 23.2 (1992), pp. 133–177.

[Wu+22] Xingjiao Wu, Luwei Xiao, Yixuan Sun, Junhang Zhang, Tianlong Ma, and Liang He. “A Survey of Human-in-the-Loop for Machine Learning”. In: *Future Generation Computer Systems* 135 (2022), pp. 364–381.

[XPT13] Petros Xanthopoulos, Panos M. Pardalos, and Theodore B. Trafalis. “Linear Discriminant Analysis”. In: *Robust Data Mining*. New York, NY: Springer New York, 2013, pp. 27–33.

[Zan19] Fabio Massimo Zanzotto. “Human-in-the-loop artificial intelligence”. In: *Journal of Artificial Intelligence Research* 64 (2019), pp. 243–252.

[ZH21] Marc-André Zöller and Marco F Huber. “Benchmark and survey of automated machine learning frameworks”. In: *Journal of artificial intelligence research* 70 (2021), pp. 409–472.

[Zha07] Jiji Zhang. “Generalized Do-Calculus with Testable Causal Assumptions”. In: *Proceedings of the Eleventh International Conference on Artificial Intelligence and Statistics*. Ed. by Marina Meila and Xiaotong Shen. Vol. 2. Proceedings of Machine Learning Research. San Juan, Puerto Rico: PMLR, Mar. 2007, pp. 667–674.

[Zhe+21] Xiawu Zheng, Yang Zhang, Sirui Hong, Huixia Li, Lang Tang, Youcheng Xiong, Jin Zhou, Yan Wang, Xiaoshuai Sun, Pengfei Zhu, Chenglin Wu, and Rongrong Ji. “Evolving Fully Automated Machine Learning via Life-Long Knowledge Anchors”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 43.9 (2021), pp. 3091–3107.

[ZJ16] Qingnan Zhou and Alec Jacobson. *Thingi10K: A Dataset of 10,000 3D-Printing Models*. `https://ten-thousand-models.appspot.com/`. Accessed: 2024-05-30. 2016.

[ZL23] Yang Zhao and Qing Liu. “Causal ML: Python Package for Causal Inference Machine Learning”. In: *SoftwareX* 21 (2023). online only `https://doi.org/10.1016/j.softx.2022.101294`.

Appendix A

Data Dictionaries

Field	Type	Description
bbox	List[Float]	Defines the extent of the geometry in 3D space.
#verts	Integer	Total count of vertices in the geometry.
#faces	Integer	Total count of faces in the geometry.
#edges	Integer	Total count of edges in the geometry.
#points	Integer	Total count of points in the geometry.
mean_curv	Float	Average curvature across the geometry.
gaus_curv	Float	Measure of curvature at a point on a surface.
max_length	Float	Maximum length among the elements in the geometry.
surfs	List[String]	Types of surfaces present in the geometry.
surface	Float	Total surface area covered by the geometry.
curves	List[String]	Types of curves present in the geometry.
#sharp	Integer	Total count of sharp elements in the geometry.
#parts	Integer	Total count of parts in the geometry.
tolerance	Float	Tolerance level of the geometry.
volume	Float	Total volume occupied by the geometry.

Table A.1: The Data Dictionary for the ABC dataset.

Field	Type	Description
file_id	Integer	Identifier for the file.
num_vertices	Integer	Total count of vertices in the geometry.
num_faces	Integer	Total count of faces in the geometry.

Field	Type	Description
num_geometrical_degenerated_faces	Integer	Number of faces that are geometrically degenerated.
num_combinatorial_degenerated_faces	Integer	Number of faces that are combinatorially degenerated.
num_connected_components	Integer	Number of connected components in the geometry.
num_boundary_edges	Integer	Number of boundary edges in the geometry.
num_duplicated_faces	Integer	Number of duplicated faces in the geometry.
euler_characteristic	Integer	Euler characteristic of the geometry.
num_self_intersections	Integer	Number of self-intersections in the geometry.
num_coplanar_intersecting_faces	Integer	Number of coplanar intersecting faces in the geometry.
vertex_manifold	Integer	Indicator of whether vertices are manifold (0 or 1).
edge_manifold	Integer	Indicator of whether edges are manifold (0 or 1).
oriented	Integer	Indicator of whether the geometry is oriented (0 or 1).
total_area	Float	Total surface area of the geometry.
min_area	Float	Minimum face area in the geometry.
p25_area	Float	25th percentile of face area distribution.

Field	Type	Description
median_area	Float	Median face area in the geometry.
p75_area	Float	75th percentile of face area distribution.
p90_area	Float	90th percentile of face area distribution.
p95_area	Float	95th percentile of face area distribution.
max_area	Float	Maximum face area in the geometry.
min_valance	Integer	Minimum vertex valence in the geometry.
p25_valance	Integer	25th percentile of vertex valence distribution.
median_valance	Integer	Median vertex valence in the geometry.
p75_valance	Integer	75th percentile of vertex valence distribution.
p90_valance	Integer	90th percentile of vertex valence distribution.
p95_valance	Integer	95th percentile of vertex valence distribution.
max_valance	Integer	Maximum vertex valence in the geometry.
min_dihedral_angle	Float	Minimum dihedral angle between adjacent faces.
p25_dihedral_angle	Float	25th percentile of dihedral angle distribution.

Field	Type	Description
median_dihedral_angle	Float	Median dihedral angle between adjacent faces.
p75_dihedral_angle	Float	75th percentile of dihedral angle distribution.
p90_dihedral_angle	Float	90th percentile of dihedral angle distribution.
p95_dihedral_angle	Float	95th percentile of dihedral angle distribution.
max_dihedral_angle	Float	Maximum dihedral angle between adjacent faces.
min_aspect_ratio	Float	Minimum aspect ratio of faces.
p25_aspect_ratio	Float	25th percentile of aspect ratio distribution.
median_aspect_ratio	Float	Median aspect ratio of faces.
p75_aspect_ratio	Float	75th percentile of aspect ratio distribution.
p90_aspect_ratio	Float	90th percentile of aspect ratio distribution.
p95_aspect_ratio	Float	95th percentile of aspect ratio distribution.
max_aspect_ratio	Float	Maximum aspect ratio of faces.
PWN	Integer	Presence of holes (0 or 1).
solid	Integer	Solidity indicator (0 or 1).
ave_area	Float	Average face area in the geometry.
ave_valance	Float	Average vertex valence in the geometry.

Field	Type	Description
ave_dihedral_angle	Float	Average dihedral angle between adjacent faces.
ave_aspect_ratio	Float	Average aspect ratio of faces.

Table A.2: The Data Dictionary for the Thingi10k dataset.