TÉCNICO LISBOA

Feature Engineering through the Exploration of Domain Knowledge

Tiago Francisco Duarte Afonso

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisor: Prof. Cláudia Martins Antunes

Examination Committee

Chairperson: Prof. Manuel Fernando Cabido Peres Lopes
Supervisor: Prof. Cláudia Martins Antunes
Member of the Committee: Prof. Rui Miguel Carrasqueiro Henriques

July 2022

Acknowledgments

Firstly, I would like to thank my supervisor Prof. Cláudia Antunes, for the weekly guidance, assistance and knowledge throughout the entire thesis process, ensuring I could do the best work possible and without whom this work would not have been possible.

I would like to thank my parents as well, for their encouragement and caring over not only the dissertation process, but my entire life. Their hard work and sacrifice has shaped and prepared me for the future. Throughout my academic journey I have never felt anything other than kindness from everyone in my family and to every single one, thank you.

I would also like to express my appreciation to my girlfriend Beatriz, also without whom any of this would not have been possible. Her love and understanding has always fueled my to do my best and try to surpass every challenge, and thanks to her, throughout the highs and lows of this work, I have never felt alone.

Last but not least, I would like to thank my friends for their support and companionship, for helping me grow as a person. For all the kidness they have given me throughout this work, I'm very thankful.

This work was supported by national funds by Fundação para a Ciência e Tecnologia (FCT) through project VizBig (PTDC/CCI-CIF/28939/2017).

Abstract

The diversification of areas where data science is present is leading to the need for more qualified scientists. To counteract this, research has shifted towards the automation of this workflow, namely with the development of frameworks for automated machine learning (AutoML). While these frameworks already bring great advancements in some aspects of the pipeline, the data preparation step continues to face great difficulties. This work proposes an algorithm that automates preparation steps and generates features using domain knowledge represented in entity-relationship diagrams, while also defining a set of operators that can be applied to distinct kinds of data. The work is validated with a case study composed of several datasets with ER models, showing improvements in model performance over existing AutoML tools such as auto-sklearn, while also having lower processing times.

Keywords

Feature Engineering; Feature Generation; AutoML; Domain Knowledge; Entity-Relationship Diagrams

Resumo

A diversificação de áreas onde a ciência de dados está presente está a levar a uma maior necessidade de cientistas qualificados. Para contrariar isto, tem existido cada vez mais pesquisa na automatização deste fluxo de trabalho, nomeadamente com o desenvolvimento de estruturas de Machine Learning automático (AutoML). Apesar destas estruturas trazerem grandes avanços em alguns aspetos do processo de ciência de dados, a fase de preparação dos dados continua a enfrentar grandes dificuldades. Este trabalho propõe um algoritmo que automatiza os passos de preparação e gera variáveis usando conhecimento de domínio representado em diagramas entidade-relação, também definindo um conjunto de operadores que podem ser aplicados a tipos distintos de dados. O trabalho é validado num caso de estudo composto por vários conjuntos de dados com modelos ER, mostrando melhorias em performance comparado com ferramentas AutoML como auto-sklearn, com tempos de processamento inferiores.

Palavras Chave

Engenharia de Variáveis; Geração de Variáveis; AutoML; Conhecimento de Domínio; Diagramas Entidade-Relação

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

AI	Artificial Intelligence
ASUM-DM	Analytics Solutions Unified Method for Data Mining
AutoML	Automated Machine Learning
CRISP-DM	Cross-Industry Standard Process for Data Mining
DANKFE	DomAiN Knowledge based Feature Engineering
DAG	Directed Acyclic Graph
DS	Data Science
EDA	Exploratory Data Analysis
ER	Entity-Relationship
FOL	First-Order Logic
JSON	JavaScript Object Notation
KDD	Knowledge Discovery in Databases
KNN	K-Nearest Neighbors
ML	Machine Learning
OWL	Web Ontology Language
PRS	Production Rule System
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SAS	Statistical Analysis System
SEMMA	Sample, Explore, Modify, Model, and Assess
XML	Extensible Markup Language

1

Introduction

Contents

1.1 Introduction

Throughout the years, the amount of data that is collected and processed has increased exponentially. Treating data manually has become intractable, which is why development in machine learning has also increased massively. In this era of Big Data, the more data that can be processed by a system, the better the information that can be retrieved for it and the more robust it can get. Processes for turning raw data into functional knowledge have been defined and refined into what is known nowadays as *Data Science*.

This growth is making data science and machine learning expand in domains, as companies and industries race to use data-driven approaches to find the best insights. This is leading to companies not having enough data scientists that have the necessary amount of experience needed to deal with this amount of data [5]. To counteract this, research is inclining towards the automation of the data science pipeline in order to be able to gather valuable insights without the need for human intervention.

These Automated Machine Learning (AutoML) tools can be a solution to the high demand and low supply of data scientists, and are already tackling important parts of the pipeline, such as model selection and hyper-parameter optimization. However, the success of machine learning algorithms does not depend solely on the quality of these algorithms, but mainly on the quality of the data preparation previously performed to the data in study. Indeed, data preparation is a crucial step of the pipeline which occupies between 70% to 80% of the time spent on the Knowledge Discovery in Databases (KDD) process, requiring a set of transformations to be applied to enhance the data feeding the learning algorithms. In this regard, current AutoML solutions still lack quality in feature engineering and in particular, feature generation, employing simple or no solutions.

One of the main reasons for this is that the frameworks try to remain domain agnostic, adopting black-box approaches (where the user is only required to export the dataset and tweak some configurations) that work for a variety of data types and datasets. In this manner, they do not allow for the exploration of available domain knowledge. Consequently, this approach to the automation of the KDD process also leads to some mistrust over AutoML methods within the data science community [6].

Recognizing that harnessing domain knowledge improves the KDD process [7], we argue that it is beneficial to represent this knowledge and explore it through automation tools to increase the amount of information that can be extracted from datasets. With this work, we propose to represent domain knowledge through Entity-Relationship (ER) diagrams, and present an algorithm, DANKFE, that automatically generates variables from them. The algorithm receives a diagram and a dataset, whose variables correspond to entities in the diagram; it then generates a new variable for each relationship described in the diagram, and fill its values for each record in the given dataset, following the description and constraints imposed in the diagram. The diversity of variables to generate only depends on the knowledge made available in the diagram, but for easiness of operation we propose a set of relationship templates that

automatically enrich a dataset, employing different types of operations. Additionally, to reap the benefits of running the dataset through the entire KDD process, we couple the DANKFE algorithm with other data preparation steps, returning a preprocessed and extended dataset and ER model which can then be used for data mining.

The work will be validated both in its efficacy and efficiency, using a case study composed of several public datasets where an ER diagram was created for each, representing domain knowledge. The baseline, preprocessed and extended versions of each dataset were evaluated both in time and performance with the best classifiers trained over each version, with a number of Machine Learning (ML) models. All versions were also compared with a popular AutoML framework, auto-sklearn [8] over the original dataset. The proposed algorithm was also studied in its scalability. Results show an improvement both in performance and in computational cost, due to the generation of useful domain-specific features, compared to methods that do not use any domain knowledge.

1.2 Organization of the Document

This thesis is structured as follows: next (chapter 2), we provide some background on the automation of the knowledge discovery process, feature generation and knowledge representation as well as give a description of the state of the art in feature generation. The problem in study and proposal are described in chapter 3, addressing the main benefits and difficulties, possible operations and the various iterations of the algorithm, which is then evaluated in chapter 4 with an additional description and profiling of the case study, as well as the methodology used to evaluate the proposal. The thesis concludes with chapter 5, where a summary of the work is presented, along with guidelines for future work and current limitations.

2

Background and Related Work

Contents

This section briefly reviews concepts regarding the KDD process, feature generation and knowledge representation, as well as showing important works regarding feature generation, either with or without the use of domain knowledge. We also pay special attention to works that work on feature generation with domain knowledge while using knowledge representation mechanisms.

2.1 Background

2.1.1 KDD Process

KDD is the predecessor term to what is nowadays called Data Science (DS). This term, created in 1989 [9], is the first scientific term to combine the entire process of extracting and using valuable information from raw data, with well-defined intermediate steps. While this process was first defined towards exhaustively exploring and revealing relationships in large databases, it has since expanded into many different areas, thus taking on the broader term of Data Science [10]. However, the overall described process remains the same no matter the context:

1. Understanding the domain in need of study, the relevant sources of knowledge that exist before investigation, and defining a goal for the process.

2. Creating or finding a target dataset or datasets to perform the KDD process.

3. Cleaning and pre-processing the data. This can be done through the use of various known strategies to remove missing values or deal with noisy data, outliers or other errors.

4. Using feature engineering techniques to know which features are considered redundant or irrelevant, increasing the expressiveness of existing features, or generating of new relevant features. While this work focuses on the entire KDD process, this step is the most relevant and will be explored with further detail below.

5. Matching the created goal in step 1 to a Data Mining approach (classification, regression, clustering, etc), depending on the context and objective previously defined.

6. Exploratory Data Analysis (EDA) to obtain more information regarding the domain, with the goal of finding interesting patterns. This profiling step includes characterization of the data according to several properties - granularity, distribution, sparsity and dimensionality. This is the step where we infer the richness of the data, possible future difficulties and define strategies on how to overcome them. Selecting the model/hypothesis on which to evaluate the data is also included in this step.

7. Data Mining, where algorithms for analysis and discovery are applied to the transformed data to reach the interesting patterns, taking into account the approach decided in step 5. These

algorithms are usually well defined and tested methods taken from the field of Machine Learning. Additional domain knowledge can be used to further improve the quality of the models, either via helping select a better model more suitable to the dataset, or by refining them by tuning parameters and hyper-parameters.

8. Interpreting the mined patterns. After the interpretation of the generated models, we can return to any of the previous steps to further improve the quality of the results.

9. Delivering / deploying the generated model. These models can be put into production, documented and then further operated and optimized. They can be classified via their simplicity (since simpler models are easier to understand and are more capable of generalization), via their certainty (the trust or confidence put into a model), or via its utility (how interesting the model is, both through coverage – the probability that the information is used, and the novelty – how unexpected the results are).

The KDD process and the Data Mining step can sometimes be mixed up, but this work uses the definition that Data Mining is a step inside the overall KDD process, and it is also the step where the most research has gone into. Since the focus of this work revolves around the generation of features by exploration of domain knowledge, it is able to follow the KDD process with a well-defined goal.

This process has also been structured and implemented in an industrial context. Cross-Industry Standard Process for Data Mining (CRISP-DM) was proposed in 1996 as a cyclic process for constant improvement of learned models, not only to identify and fill flaws in the different stages, but also to create a framework for finding the best possible models as easily as possible.

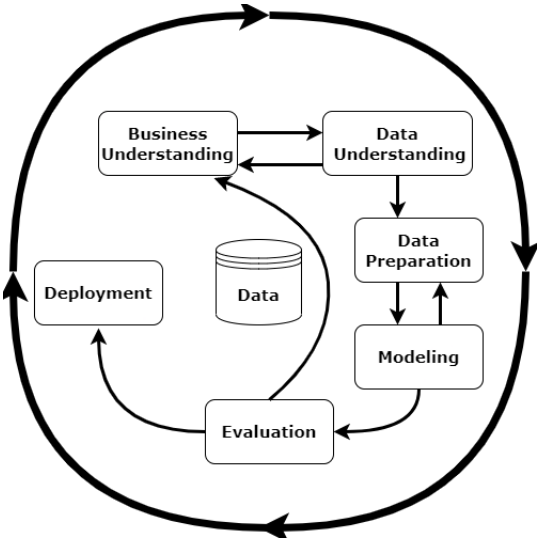

Figure 2.1: The CRISP-DM Process.

While CRISP-DM does not enumerate as many steps as the KDD Process in [9], the underlying

principles are the same, showing a cyclic process in project management of a data mining process. This model, which is still one of the most frequent methodologies for Data Science workflow [11], is divided into six phases [12, 13]:

1. Business Understanding - Similar to all projects, the first step is understanding the needs and objectives, determining the most appropriate goals for data mining and producing a plan, which later can be refined. This step is similar to steps 1 (domain understanding) and 5 (goal creation) of the KDD process.

2. Data Understanding - collection and creation of the data, followed by its thorough examination, identifying potential patterns or issues that can arise. This relates to steps 2 (data collection) and 6 (exploratory data analysis) of the KDD process.

3. Data Preparation - data cleaning, preprocessing and feature engineering to extract the true potential of the dataset, eliminating noise, missing values, redundant or irrelevant features, or constructing new ones, either with or without domain knowledge if available. This is related to steps 3 (data preprocessing) and 4 (feature engineering) of the KDD process. This is the most harduous and subjective step of the process, and therefore the one that usually takes the longest time.

4. Modeling - selecting the best suited machine learning model to test in the dataset, fitting it to the data either through domain knowledge or through the performance of the model. This step is similar to step 7 (data mining) of the KDD Process.

5. Evaluation - interpretation of the model results and determining whether it is better to roll back to previous steps and improve the model (either by changing or optimizing it), or if it is already strong enough to be deployed. This relates to step 8 (interpretation) of the KDD process.

6. Deployment - Monitoring, documenting and extracting the benefits of the model in a professional setting. Model optimization may still be required even after it is put into production. This relates to step 9 (deployment) of the KDD process.

While the outcome of each phase depends on the previous, the process does not end when the model is deployed. Rather, the lessons and obstacles that were overcome initiate new business questions, enriched with the knowledge obtained from previous mining processes.

Other famous methodologies for describing and organizing the Data Science process are Analytics Solutions Unified Method for Data Mining (ASUM-DM) [14], an extension of CRISP-DM introduced by IBM in 2015, which combines the Agile approach to the Analytics workflow via an iterative and incremental development of the solution, and Sample, Explore, Modify, Model, and Assess (SEMMA) [15], developed by the Statistical Analysis System (SAS) Institute. SEMMA is a functional toolset composed

of the 5 stages that make up its name, which are roughly equivalent to the KDD process phases [16], but are tailored to the SAS Enterprise Miner software developed by the company.

2.1.2 Feature Generation

As stated in section 2.1.1, one of the most important and time-consuming steps [2] in the KDD process is step 4, where given a defined objective and a dataset to work with (and some possible problems in the dataset have already been addressed via preprocessing), it is then possible to work closely with the data at hand to explore which features are important or which can be considered redundant, if our existing features need to provide more information, or if creating new features (either based on existing data or other sources) can help improve the quality of the future models. This step is known as *Feature Engineering*, and it is a fundamental step in ensuring that our model is able to achieve strong results, seeing that the input features are the key to achieving the best model results further down the pipeline [17].

Firstly, we can define *feature* as an attribute of the data that is meaningful to the problem we are facing. Features are usually represented as n-dimensional vectors, which are often numerical depending on the context to which they are being put to use. Since features are represented as a vector, we can consider each feature to be a different dimension and map our data to a *feature space*. Throughout the iterative KDD process, the dimensionality of this feature space can change, either increasing: due to found relationships between features that at first glance seem irrelevant to the problem, but through some operation become much more powerful, or decreasing: to avoid problems such as the curse of dimensionality, where even with a large dataset, the data becomes too sparse on the feature space and most of it becomes unknown, which in turn can greatly diminish the performance of machine learning algorithms [18].

The different techniques in feature engineering can be divided into three categories:

- *Feature selection* - the process of identifying redundant (relevant but can be removed due to the presence of another feature that provides the same information) and irrelevant (bring no information when discriminating between classes) features. It is known that by having many variables that correlate with each other, there is not much knowledge that can be retrieved from them (as they all convey similar information) [17]. Also, having too many features and not enough observations leads to overfitting and high variance [19]. Therefore, a series of techniques are required to reduce data dimensionality and only keep the non-redundant and relevant variables to describe the data.

- *Feature Extraction* - the process of using a method to extract a set of new features from the original ones [20]. Techniques in feature extraction usually differ from feature selection by being mostly unsupervised, which means they can always be applied during the data preparation phase

regardless of the task. The newly created (usually orthogonal) variables are a linear or non-linear transformation (usually a reduction) from the original feature space to a different one. However, a downside to this method is that the new variables lose comprehensibility.

- *Feature Generation* or *Construction* - the process of creating new variables from the original ones. This can be done with or without the use of domain knowledge, which will be seen later. Unlike feature extraction, feature generation analyzes relations among features, augmenting the feature space [20]. These techniques are limited to applying operations on subsets of variables, with the choice of which operations being either *instance-based* (use of probability or information theory measures in an automatic fashion), *hypothesis-based* (ranking features according to learned hypotheses) or *knowledge-based* (using existing domain knowledge). [3]. The domain knowledge used to improve a model's induction also does not need to be complete, as it is proven that fragmentary knowledge can still be applied for searching new features, changing the shape of the feature space [21].

Data scientists usually employ a combination of these processes to get the most out of their data. Even though there is a lot of literature regarding optimizations and methods for selecting and extracting features, since the theme of this work is feature generation, it will be the category inside feature engineering that will be explored in depth in the following sections.

But before going there, since in order to use domain knowledge we require some method to represent it, we need to address the field of knowledge representation, briefly reviewing its main approaches and properties used for supporting the data science process.

2.1.3 Knowledge Representation

Knowledge Representation is a branch of Artificial Intelligence (AI) which focuses on "how knowledge can be represented symbolically and manipulated in an automated way by reasoning programs" [22]. According to [23], knowledge representation can have 5 roles:

- As a surrogate for the entity or event it is representing.
- As a set of ontological commitments, where we decide what is relevant to represent and what is not, since "All representations are imperfect, and any imperfection can be a source of error" [23].
- As a fragmentary theory of intelligent reasoning, providing the inferences that it allows or recommends.
- As a medium for efficient computation, meaning that every representation has a trade-off between expressiveness and computational performance.

- As a medium of human expression, meaning that there is also a trade-off between expressiveness and easiness of communication.

As we can see, there are trade-offs when combining the way we represent knowledge and how we reason with it. Some formalisms are very strong for representing the information derived from data, but this comes at a cost in performance and efficiency (or having to deal with problems such as undecidability). Choosing the right way to represent knowledge depends on the context and task it is going to be used for. Some of the ways to represent knowledge are:

- *Logic-based languages*. First-Order Logic (FOL), also known as Predicate Logic only has a few basic symbols but it has enormous expressiveness, still being highly-used to this day. However, for practical day-to-day AI, it is hard to implement due to the difficulty in finding formulas that are always true, and that can capture a lot of knowledge at the same time [24]. Other kinds of logic are *Propositional Logic* (uses no quantifiers), *Modal Logic* (has different modalities such as possibility, necessity, knowledge, belief and perception) [25], *Markov Logic* (which unifies FOL with probabilistic models coming from the statistical part of AI, in the form of Markov Logic Networks [24]) and Higher-order Logics (which can have more quantifiers or stronger semantics, again trading off performance for expressiveness) [26].

- *Production Rule Systems*, which are good for procedural knowledge, based on well-founded rules that are applicable when a condition is met. They are composed of a sensor, which detects when the condition is activated (`if`) and an action, which triggers the result (`then`). [22].

- *Object-oriented representations*, which unlike the previous ones have hierarchies, organizing knowledge into objects. The most famous example are frames [27], which work similarly to how our memory processes events, having fixed top levels and lower-level ”slots” where the specific data is kept.

- *Semantic Networks*, which are directed graphs that represent objects/entities (nodes) and their semantic relations (arcs) [28].

2.1.3.A Ontologies

As stated before, logic representations can be highly expressive even when using few symbols. This is due to the predicates (properties or relations) that are attached to them. This set of predicates can be either domain-dependent or independent and are part of an *ontology* which aggregates all relevant things for the domain at hand. The sets of predicates represent the ontological commitments that are part of the knowledge representation [29].

Extending frames and semantic networks [22], ontologies are a ”representation vocabulary, often specialized to some domain or subject matter” [30], being also described as a ”body of knowledge describing some domain (...) using a representation vocabulary”. The vocabulary used are the terms that describe the facts, and the collection of facts about that domain is known as the knowledge body. Ontologies are helpful for every step of the KDD process, being used for inspection of the domain while choosing the task to operate (Business Understanding), mapping the elements of the data schema to the knowledge base tailored for our needs (Data Understanding), identifying how to group attributes semantically (Data Preparation), eliminating hypotheses that make no sense semantically (Modelling), interpreting the results with the structured knowledge that is part of the ontology (Evaluation), and aiding in the integration of new knowledge (Deployment) [31]. However, they are only helpful if querying through them to obtain the relevant knowledge is done quickly and efficiently [32].

Ontologies are a fundamental part of the *Semantic Web*, an extension of the World Wide Web whose objective is to make the data available online machine-readable. As a portable way of representing knowledge, ontologies are used in encoding semantic information, with technologies such as Resource Description Framework (RDF) and Web Ontology Language (OWL). Huge knowledge bases are continuously created and used for discovering and integrating more information from the data that is constantly being added to the internet. [33, 34]. Nowadays, the Semantic Web has also expanded to linked data (information as graphs) and knowledge graphs (which differ in consistency and are more for professional/industrial use) [35], however ontologies still remain important as a representation and as schemes for the graphs.

Ontologies work similarly to schemes in database systems, with the core difference being that conceptual schemes describe the data and its relationships, and ontologies describe the knowledge derived from the data [36]. An **Entity-Relationship Model** is a conceptual ontology that expresses how entities of interest are related in a specific domain of knowledge. These data models are usually implemented in relational databases, incorporating important semantic information about the real world [37].

Lastly, another form of knowledge representation are **Bayesian Networks**, which similarly to semantic networks are Directed Acyclic Graphs (DAGs) that represent variables and their dependencies. Each node has a conditional probability distribution of the outcomes of the child node in condition to the combinations of the parent nodes [38]. This model is best suited when knowledge of prior events is useful for evaluating models, and allows for a subset of the variables to be conditionally independent. Inference in Bayesian Networks is trying to find out the probability distribution of a variable based on the observations of other variables, calculating the posterior probability $P(X = x|evidence)$. Various mechanisms can be applied for inference [39]. If we define formulas for the states of a node and its parents in a Bayesian Network, we can also represent it as a Markov Logic Network, which is an extension of first-order logic with the addition of a probability for each world, by adding weights to the formulas (the more worlds

it violates in the Knowledge Base, the less probable it becomes). More information on Markov Logic Networks can be seen at [40].

As stated before, the choice of knowledge representation method depends strongly on the context, domain knowledge availability and need for expressiveness. For smaller domains, more expressive methods can be used as their performance will not become a hindrance. Larger domains can benefit from more complex representations such as ontologies or higher-order logic. The representation best suited for this work will be discussed in the following sections.

2.2 Related Work

This section highlights the work that has been done throughout the years regarding feature generation and the different methods that can be applied to this process.

As stated in section 2.1.2, feature engineering (and therefore generation) is usually the most time-consuming step of the KDD process [2], especially since it requires human interaction and intuition to obtain the best results. This can be subjective, costly and limits the process' repeatability, which sometimes makes it harder for adoption in the real world (where there is a trade-off between performance/optimization and applicability/time to market) [41]. To counteract this, there have been numerous works on automating the generation of features, either with or without the use of domain knowledge to improve induction.

2.2.1 Feature Generation without Domain Knowledge

Feature Generation does not necessarily need domain knowledge in order to be applied. It can follow a data-driven approach that works either by applying operators to features (unary transformations), or by combining features of the same data types with certain operators (binary or n-ary transformations or aggregations). Examples of unary transformations are logarithm, exponential, square, square root, absolute value, extraction of certain parts of the value (such as day, month, year for timestamps). Examples of binary transformations are sum, product, difference, division, etc. Aggregations with two or more columns can use operators such as count, max, min, average. The type of operator depends on the data type. Operators like Cartesian products and Boolean expressions like M-of-N and X-of-N (boolean threshold functions - for example 2-of(A,B,C) is equal to the AB + AC + BC) are better for nominal features, while for numerical features operators like the ones mentioned before can be used.

Historically, the first feature generation methods that appeared in research focused on these data-driven (only uses the input data for guidance) or hypothesis-driven (the induced hypotheses are used for guidance) methods. *Pagallo and Haussler* [42] developed a system called FRINGE that generates new features as Boolean combinations (conjunctions and disjunctions) of existing features, creating new

decision trees until the variable set can no longer be increased. *Murphy and Pazzani* [43] developed a similar system, also for decision trees, that created M-of-N concepts. These operations are also used by *Zheng* [44]. These methods are hypothesis-driven because the features are based on the hypotheses generated by the decision trees (and do not use any specific measure for evaluation). *Ravagan et al.* [45] and *Hu and Kibler* [46] use data-driven approaches instead, that do not depend directly on previous trees and instead use metrics like Information Gain. Both use Boolean operations (conjunction and negation) with the first returning a decision tree with the combinations of the features with higher gain iteratively generated combined with original features, and the latter working in a similar fashion but returning the feature set.

To overcome the limited amount of operators used for feature generation, *Markovitch and Rosenstein* [1] created a flexible framework to describe feature generation algorithms. By providing a set of constructor functions and a dataset, the FICUS algorithm builds decision trees with the combination of original and generated features (that come from the constructor functions defined by the user). These generated features can then be used inside the new decision trees. To deal with the generation of irrelevant features, a feature selection method is used to filter irrelevant variables (since a common problem with all these approaches is the massive increase in features). This selection can either be done by a data-driven function (which calculates complexity and improvement of the features to maintain the search as simple as possible) or an hypothesis-driven function (which evaluates the contribution of each feature in the decision tree hypothesis). After the generation, features are further filtered using information gain to ensure relevancy. While the method of action is similar to the aforementioned algorithms, it is more flexible as it can use any kind of input (which can be dynamic) and only exploits appropriate constructor functions.

The work of *Fan et al.* [47] also generates features using decision trees. The algorithm uses a divide-and-conquer strategy where a new feature set is created at each node (partitions of the data), by selecting a random weighted operator, which is then tested using information gain. When the best feature is chosen, it is added to the set and the weights are updated. The algorithm continues recursively until it reaches a stop condition. This approach avoids exhaustive searches in the feature space while also avoiding the need for domain knowledge.

Kanter and Veeramachaneni [48] introduced the Data Science Machine in 2015, a system capable of automating feature engineering in relational databases. It uses an algorithm called *Deep Feature Synthesis*, which follows the relationships between entities, creating 3 new types of features: entity features (unary operation), direct features on one-to-one relationships, and relational features on one-to-many relationships. The operations used are aggregation SQL operations such as average, sum, count, etc. This algorithm extensively searches and enumerates all possible features, performing feature selection (which can be slow due to the amount of new features) and hyper-parameter optimization

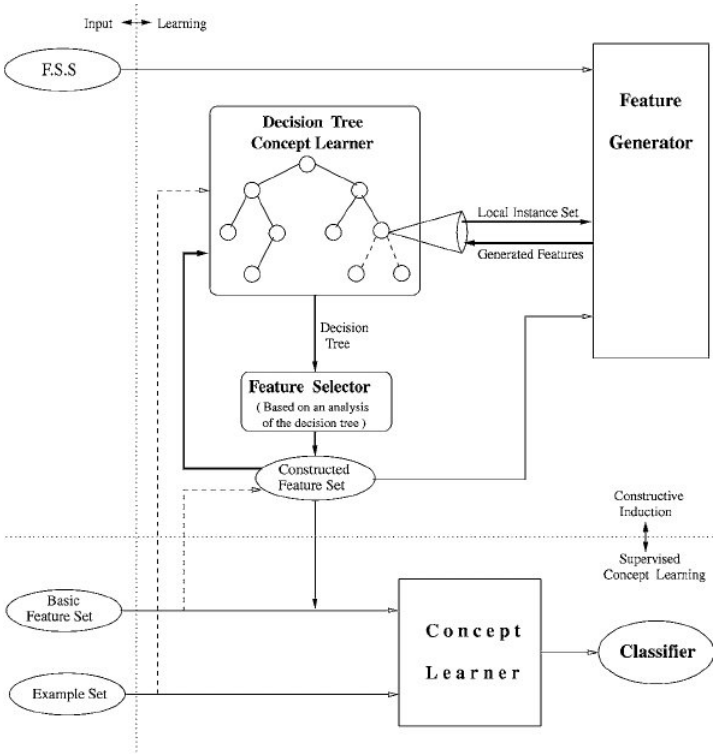

Figure 2.2: The FICUS algorithm, from *Markovitch and Rosenstein* (2004) [1]

afterwards. This approach also does not work for unstructured data such as text or sequences.

This ”expand-reduce” method was further implemented by *Katz et al.* [49], *Lam et al.* [50] and *Kaul et al.* [51]. ExploreKit [49] is able to work on non-relational data by generating a large number of features created from common operators, ranking them using a scoring function based on *meta-features* (characteristics of the dataset that affect the effectiveness of a feature) and then selecting only the ones considered relevant. Due to the large amount of features, it takes an intractable amount of time to run. OneBM [50] works on unstructured data such as text or sequences (complementing the work of *Kanter and Veeramachaneni* [48]) but inside databases, by linking relationships, joining tables and running a depth-first search to perform a set of transformation functions on the features, selecting only the relevant ones afterwards. AutoLearn [51] associates pairs of features, first by filtering irrelevant ones with Information Gain, followed by calculating the correlation between pairs. For the relevant pairs, it constructs features using regularized regression techniques, further filtering redundant and irrelevant pairs with stability selection and information gain respectively. This approach creates less intermediate features than previous works.

Other methods for feature generation that do not make use of domain knowledge are:

- Hierarchical greedy search, used by *Khurana et al.* [41], exploring a transformation graph in a greedy way and applying the transformations to the features, which also allows for the composition

of transformations, whilst avoiding more exhaustive searches.

- Meta-Learning with Neural Networks, used by *Nargesian et al.* [52], where neural networks are trained with a collection of datasets. Based on this past experience and the meta-features of the current dataset, the algorithm proposes useful transformations to increase the feature set.
- Reinforcement Learning, used by *Khurana and Samulowitz* [53], in the APRL system, which trains an agent to either perform feature transformations (with predefined functions), or model building and hyper-parameter optimization, given a budget constraint.
- Other types of Neural Networks, used by *Chen et al.* [54] and *Xie et al.* [55]. The former uses a Recurrent Neural Network that transforms the features using transformer functions, with a controller that uses reinforcement learning, and the latter using a Graph Neural Network and an adjacency tensor for the features, increasing the feature set with this tensor.
- Genetic Programming, used by *Krawiec* [56], *Smith and Bull* [57] and more recently *Tran et al.* [58] and *Ma and Gao* [59], which use a variety of methods for generating features that enrich datasets. Genetic programming has the advantage of being highly flexible and known to get good results in classification problems, but it requires a lot of parameters and computational cost to achieve them.

More recently, due to the increasing amount of areas where machine learning is present, the need for developers and expertise became bigger than ever. This led to a shift in research to the automation from just feature engineering to more parts of the KDD process (such as model selection and hyper-parameter optimization, as seen in fig. 2.3). This led to a huge increase in AutoML frameworks in recent years, all of them having the goal of returning the best approach for a dataset with as little human intervention as possible [60].

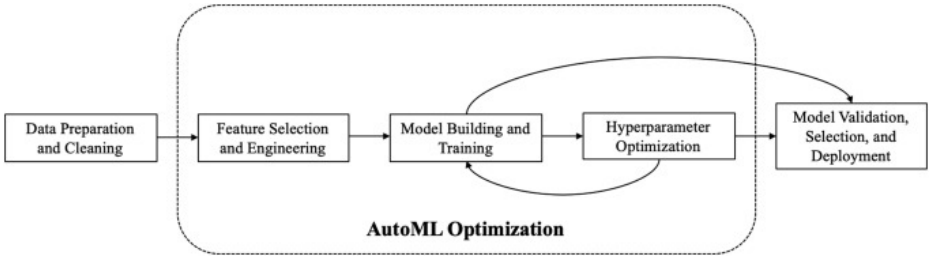

Figure 2.3: The usual AutoML pipeline, from *Waring et al.* [2]

While much can be said about what AutoML frameworks exist, what problems do they tackle in the pipeline and how they tackle them [61, 62], this work only focuses on the feature generation step, where the most known frameworks still apply only a few techniques at best. Frameworks such as Auto-WEKA [63], Auto-Keras [64] and Auto-Pytorch [65] do not apply any feature generation techniques, while Auto-sklearn [8] uses embeddings, clustering, matrix decomposition and one-hot encoding, as

well as meta-features. Auto-Gluon [66] only uses simple data preprocessing techniques, as well as H2O [67]. TPOT [68] uses meta-features and is the only framework mentioned that uses polynomial combinations. We can see that feature generation is an area where AutoML still has much room to improve, especially with the inclusion of techniques that make use of domain knowledge, which could help improve adoption, as well as improve results, since these frameworks being black-box systems still leave users skeptic whether it is best to incorporate domain knowledge into their own techniques or just let them run automatically, without knowing the processes that were applied on the data [6].

2.2.2 Feature Generation with Domain Knowledge

Several works have been published throughout the years researching the incorporation of domain knowledge into feature generation. This knowledge does not need to be a set of whole theories. Even if it is fragmented, it can help narrow down the feature space. *Donoho and Rendell*'s work [21] investigates the different kinds of domain knowledge and their influence on feature generation.

CITRE [3] is an hypothesis-driven algorithm similar to FRINGE that uses Boolean operators to iteratively find patterns in a decision tree, constructing and testing combinations of features. After the expansion in feature space, the algorithm uses domain knowledge to filter out undesired features. This knowledge however is inside the algorithm and not in the representation, and it is not able to work with partial knowledge. The use of domain knowledge in CITRE can be seen in Figure 2.4. A similar approach is followed by *Aha* [69], which creates conjunctions of features that match positive instances and do not match negative ones. It also uses domain knowledge to filter irrelevant variables, but the amount of features it creates is computationally impossible to scale.

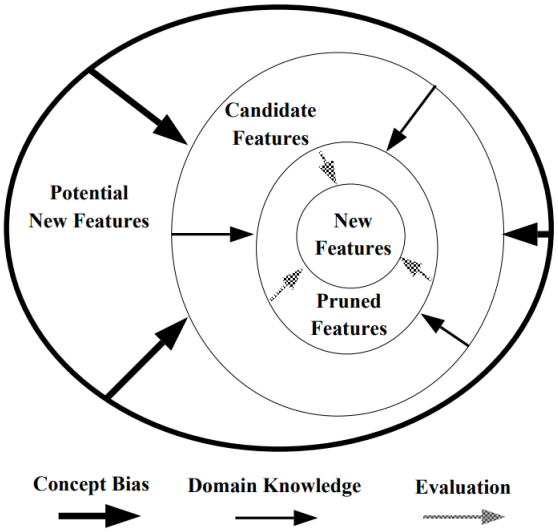

Figure 2.4: Variable filtering in CITRE, from *Matheus and Rendell* [3]

The FICUS algorithm [1], can also be considered an algorithm that uses domain knowledge since it

lets the user choose which constructor functions are used in the generation step. The framework allows users to exploit this knowledge to filter out methods that are not relevant to the task at hand, using combination methods for generating features, as explained in section 2.2.1.

When working with relational data, the work of *Aronis and Provost* [70] uses existing domain knowledge to create relational terms, by putting the training set and knowledge into an inheritance hierarchy and using formula propagation techniques to find relations. *Badian and Markovitch* [71] present an algorithm that increases the feature space by joining common features between the dataset and other datasets that are used as knowledge sources, following an ”expand and reduce” approach.

There are also examples of feature generation in other domains. Regarding textual data, the works of *Gabrilovich and Markovitch* [72], *Wang et al.* [73], *Hu et al.* [74] and *Zhang et al.* [75] use Wikipedia as a knowledge base for enriching datasets for text classification or other NLP tasks, either via measures such as Bag of Words and the amount of re-directions, or using semantic relations. While these approaches insert a lot of domain knowledge into the datasets, which improves the performance of models, it also brings a lot of noise which needs to be filtered. The Word2Vec algorithm [76] also generates features as vectors, using domain knowledge provided from large corpora.

2.2.2.A Feature Generation based on Knowledge Representation Formalisms

Features can not only be generated with the use of domain knowledge, as explained, but also through the exploration of a specific knowledge representation formalism.

Cheng et al. [77] created a graph-based language for feature generation in linked data, by querying the relations inside the data. This language framework allows extraction of information from knowledge bases such as YAGO and DBPedia. Similarly, the work of *Paulheim et al.* [78] proposed extensions to the Linked Open Data paradigm (best practices for linked data in the Semantic Web [79]), with operations for adding features to data also based on linked data sources. Both these approaches are forms of propositionalization, transforming the structured problems into a simpler format that can be input to the usual data mining algorithms. These works are considered to follow a knowledge representation since linked data is composed of resources which follow a schema or ontology. Approaches for propositionalization are studied in *Kramer et al.* [80].

A similar approach is the algorithm proposed by *Terziev* [4], which employs a decision tree where features are searched breadth-first, adding the relationships present in the ontology and using information gain as an evaluation rule on the new features. The features considered are outgoing paths from the origin entity, and it explores the ontology via its graph structure. The algorithm’s feature generation and exploration of ontologies is shown in Figure 2.5.

The system by *Galhotra et at.* [81] uses several forms of structured knowledge (graphs, web tables, linked data), available by an indexing system. Useful features are searched in the sources, increasing

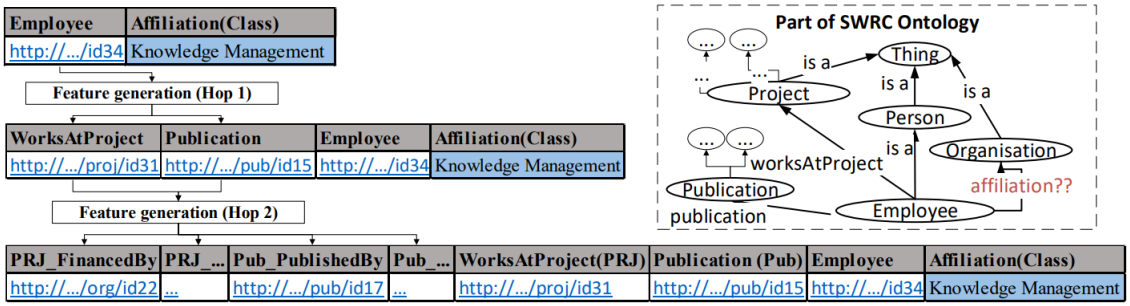

Figure 2.5: Feature Generation and Ontology exploration in *Terziev* [4]

the amount of features by linking related data. It is presented as a Python library which also shows statistics from the features and the relations between the knowledge bases and the data provided by the user.

Bloehdorn and Hotho [82] created an approach for collecting domain knowledge from known ontologies in order to improve text classification results, by using a combination of ontologies, finding candidate terms that match the dataset (as well as NLP techniques) to increase the feature space. *Friedman and Markovitch* [83] also approach this problem, by creating a flexible algorithm that given a knowledge base, it recursively adds new features to the dataset via relations in that knowledge base, until a certain depth is reached. It also proposes a divide-and-conquer algorithm for gathering knowledge into the dataset by checking local contexts (subsets of data). This algorithm was also tested for text classification problems.

Lastly, the work of *Salguero et al.* [84] proposes a methodology that also generates features from ontologies, through the combination of concepts present in the knowledge base, using a group of extension functions, either predefined or created by the user. To avoid the exponential growth in features, only certain class expressions are used, scored via the depth of their trees, which gives more importance to simpler expressions. The features are further selected by only adding to the dataset the ones that are relevant to the problem at hand (especially important if the ontology tackles many domains). This algorithm was tested in smart sensor and daily living problems, increasing the accuracy of classifiers when the ontology is structured and there is no clear idea of what the relevant features are.

3

Solution Proposal

Contents

In this chapter, we present our proposal to mitigate the problem of current automatic machine learning frameworks not focusing their attention on the generation of potentially interesting features, and therefore not benefiting from the use of domain knowledge which can improve the quality and useful information that can be retrieved from the dataset.

3.1 Problem Statement

As seen in section 2.2.2, the desire of exploring domain knowledge across the KDD process is not new, and has been pursued since its origins [7]. However, the different approaches proposed along time did not grasp enough quality to be generally adopted. We can distinguish two main approaches:

- Embedding the domain knowledge in the mining algorithms themselves. While these algorithms are undoubtedly more effective since the knowledge is directly linked with the algorithm, they suffer the downside of not generalizing well, requiring a different algorithm for each problem, making it difficult to adapt to new situations.

- Using general algorithms which explore external knowledge sources. This approach is much easier to generalize for multiple problems, but depends on the availability of knowledge bases and the need for them to be expressive enough to represent the domain expertise.

Since our problem is to incorporate the use of domain knowledge for automatic machine learning, which strives to be as general as possible (through the use of domain agnostic methods), the best approach is the latter, creating an algorithm capable of exploring external knowledge and using it to augment the feature space. Having chosen an approach, we now need to choose a way to represent knowledge. This knowledge shall be represented as specific attributes and relationships among the concepts in the domain, which depending on their properties, can lead to the creation of new variables.

After studying in section 2.1.3 possible frameworks and formalisms for representing domain knowledge and their advantages and disadvantages, we need to now choose a possible formalism that is suitable for the problem at hand. One possible approach is, by using an ontology as the means to represent knowledge, to extend the dataset by populating the variables through the application of the axioms present in the ontology. However, this approach brings a few problems. First, there needs to be a clear definition of how the axioms are matched to the dataset. While ontologies are the most expressive formalisms, their definition for each domain require significant efforts to take advantage of all elements. Second, many domains or datasets are far from being semantically rich, not having ontologies that are ready for use. While this availability continues to be a mirage, a more available alternative is to make use of databases instead of knowledge bases.

Databases are indeed the most usual data sources, and they are usually designed through ER diagrams, formalized as relational schema *a posteriori*. ER diagrams have three main elements: rectangles used to define concepts, named *entities*, ellipses for *attributes* and diamonds for *relationships* among concepts. Since these diagrams represent the majority of the elements expressible through ontologies, we may say they are simplifications of those formalisms. As a matter of fact, ER diagrams are just not expressive enough to represent axioms. Nevertheless, they are frequently used in database design which guarantees their availability for a large number of situations, with plenty of experts which are able to design them.

Having studied the best approach for the algorithm and the method for representing the knowledge, we can now present the problem statement in this new context:

Given a dataset and a corresponding Entity-Relationship diagram, create a new dataset by transforming and extending the original one, through the exploration of the knowledge expressed in the diagram.

Before proceeding with the algorithm analysis, we need to specify how the ER diagram and the dataset are related to each other. Since a dataset is solely described by a set of variables, the diagram needs to be able to represent those variables. The problem arises in choosing how to represent them, which can be done either through the entities themselves or through attributes that characterize the entities. This is known as the *reification problem* [85], and since we want to be able to manipulate the existing variables to create new ones, we choose to represent every variable as an entity, in order to reason and talk about them. In this manner, the ER diagrams have to represent all existing variables as entities, and since new variables result from the combination of existing ones, they have to be represented through relationships.

We are now ready to define an ER diagram in our context:

Definition 1. *An ER diagram is a tuple $\mathcal{KB}$=($\mathcal{E}$, $\mathcal{R}$), where $\mathcal{E}$ is the set of $\texttt{entities}$ and $\mathcal{R}$ is the set of $\texttt{relationships}$ among the entities in $\mathcal{E}$.*

Moreover,

Definition 2. *Given an ER diagram, $\mathcal{KB} = (\mathcal{E}, \mathcal{R})$ as defined before, and a dataset $\mathcal{D}$ described by a set of d variables, $\mathcal{F} = \{v_1, ..., v_d\}$: $\forall v \in \mathcal{F}\ \exists e \in \mathcal{E}$: e corresponds to v.*

In order to explore such diagrams, we translate them into JavaScript Object Notation (JSON) files, following a predefined structure. JSON is a standard text-based format for storing and transmitting structured data, used in plenty of web applications. Other formats could be used, including Extensible Markup Language (XML), Resource Description Framework Schema (RDFS) and Web Ontology Language (OWL), to name a few. Independently of the choice, the specification of the ER elements must follow a strict definition.

Each entity in an ER diagram is characterised by its *name*, used as an identifier, its *type* to help on

determining the possible operations to perform over it and a *description* optionally used to clarify any additional information about the entity.

As relationships specify the new variables to generate, they have a more extensive definition. Each *relationship* in an ER diagram is characterised by its *name* again used as an identifier, but now also used for naming the new variable to generate, *inputs* for specifying the list of entities that make up the relationship, *operations* corresponding to the sequence of operations to perform over its inputs to generate the new variable and the *constraints* its inputs have to satisfy to make the generation possible. Optionally, it may include a *groupby* parameter specifying the variable along with an aggregation may be made and *condition* for specifying which records to aggregate.

3.1.1 Operations

With this schema, since the operation or operations used to generate a new variable are specified inside the diagram, the complication of understanding each operation arrives. In order to be possible for the algorithm to understand and generate variables with all imaginable operations, a decoder would be needed inside the algorithm, that would ideally translate the operation specified into a function. Unfortunately, this would add enormous processing time and complexity to the solution. To mitigate this, we propose a set of possible operations types that can be used to generate new variables. These types work as ”relationship templates” that the algorithm is able to interpret and calculate, therefore creating new variables for the dataset.

These types of operations span a large range of possible operations, which can be specified in the ER diagram. We propose the following types:

- Decomposition operations: any operation over a single record, described by a single variable, that extracts some component from its value. Examples of these operations are the decomposition of a date into its components (*year*, *month*, *day*), decomposition of strings that follow certain patterns (*firstname*, *surname*), among others.

- Algebraic operations: any mathematical operation over a single record, described by one or more variables. Examples of such operations for a single variable are *absolute, square root*, *division, logarithm* for two variables, and *sum, product* for any number of variables.

- Mapping operations: any operation over a single record, that maps the value in one variable to another value, possibly from different types. Examples of these operations are mapping if a date is a *holiday* or which *weekday* it is. Comparing the value of a variable against some threshold or if it is equal/different from another value can also be considered a mapping operation.

- Aggregation operations: any operation to be applied over a set of records. Examples of these

operations are *sum, average, max, stdev* applied over a set of records, that satisfy some imposed condition similar to the ones achieved with a *GROUPBY* clause in an *SQL* query.

- Composition operations: a sequence of operations to be applied one after the other, as a mathematical composition of functions. This allows for multiple different operations to be applied for the generation of a variable. An example is extracting the *nr_months* that have passed from two dates, first by subtracting the two dates, which may return the number of days between them, and taking that result, converting it into the number of months.

From these operation types, two different procedures can be distinguished - the generation of variables with or without aggregations, as specified by the *groupby* parameter. We call *aggregation-based* generation the first one, and *record-based* generation the second approach. It is useful to differentiate both approaches since decomposition, algebraic and mapping operations do not require information about any other record besides the one where the operation is being applied, while aggregation operations require information about other records in which the aggregation is to be made.

Both procedures can be defined using the previous logic. Considering $\mathcal{D}$ to be a dataset, $\mathcal{F} = \{v_1, ..., v_d\}$ the set of d variables describing $\mathcal{D}$ and $\mathcal{KB}$=($\mathcal{E}$, $\mathcal{R}$) an ER diagram, as defined before.

Definition 3. *Let $r = (\Theta, \Pi, \Psi, \emptyset, null)$ be a `record-based` relationship in $\mathcal{R}$, with $\Theta \subset \mathcal{E}$ the set of `input` variables, Π the sequence of `operations`, and Ψ the set of `constraints` to satisfy.*
The procedure generates a new variable v_r, and each record $x = x_1...x_d$ in the dataset $\mathcal{D}$ becomes $x' = x_1...x_d, x_r$, with x_r filled as follows:

1. *if $\exists\theta \in \Theta\ \exists\psi \in \Psi$: $x_\theta \not\models \psi$, a $null$ value is assigned to x_r;*
2. *otherwise,*
   (a) *x_r becomes $\pi(x_{\theta_1}...x_{\theta_k})$, where π is the last operation in Π and $(x_{\theta_1}...x_{\theta_k})$ is the projection of x along each variable $\theta_i \in \Theta$;*
   (b) *if $|\Pi| > 1$ then x_r becomes $\pi_i(x_r)$ with π_i being each one of the i^th with $0 < i < j$, and j the number of operations in Π, from the $(j-1)^th$ to the first one.*

It is also possible to define *aggregation-based* procedures in a similar way:

Definition 4. *Let $r = (\Theta, \Pi, \Psi, \Delta, \phi)$ be a `aggregation-based` relationship in $\mathcal{R}$, with $\Theta \subset \mathcal{E}$ the set of `input` variables, Π the sequence of `operations`, Ψ the set of `constraints` to satisfy, Δ the set of variables to specify the `aggregation` and ϕ the `condition` to constraint the aggregation.*
The procedure generates a new variable v_r, and each record $x = x_1...x_d$ in the dataset $\mathcal{D}$ becomes $x' = x_1...x_d, x_r$, with x_r filled as follows:

1. *if $\exists\theta \in \Theta\ \exists\psi \in \Psi$: $x_\theta \not\models \psi$, a $null$ value is assigned to x_r;*

2. *otherwise,*

   (a) *a temporary variable* γ *is created for storing the projections of each* x *along each variable* $\theta_i \in \Theta$ $(x_{\theta_1}...x_{\theta_k})$ *for all records in* $\mathcal{D}'$ *satisfying the condition* ϕ*, and aggregated according to all variables* $\delta \in \Delta$*;*

   (b) *then* x_r *becomes* $\pi(\gamma)$*, where* π *is the last operation in* Π*;*

   (c) *if* $|\Pi| > 1$ *then* x_r *becomes* $\pi_i(x_r)$ *with* π_i *being each one of the* i^th *with* $0 < i < j$*, and* j *the number of operations in* Π*, from the* $(j-1)^th$ *to the first one.*

In this way, variables that require aggregation operations can be generated via an *aggregation-based* procedure, and all other specified operations can be generated via a *record-based* procedure.

3.1.2 Illustration

In order to better understand the algorithm proposed, we can consider as example the knowledge base represented by the ER diagram represented in fig. 3.1.

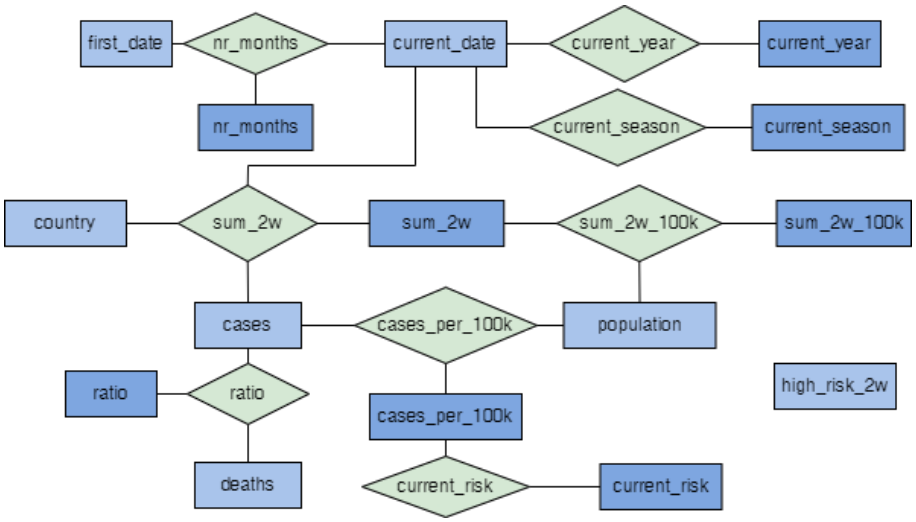

Figure 3.1: Example of the ER diagram for feature generation.

Additionally, we also consider the data on table 3.1, which corresponds as the input dataset $\mathcal{D}$ to the algorithm, which is described by the set of variables $\mathcal{F}$=$\{current\_date, cases, deaths, country, population, first\_date, high\_risk\_2w\}$.

We can see that all variables in $\mathcal{F}$ are represented in the ER diagram as entities (light blue rectangles). Besides the entities that map to each variable, there are also eight relationships (green diamonds) and eight additional entities (dark blue rectangles), which correspond to the variables that will be generated by the algorithm. Each relationship is linked to a set of entities, where the lighter ones correspond to the inputs, and the darker ones to the output (the variable that will be generated).

current_date	cases	deaths	country	population	first_date	high_risk_2w
2021/02/23	1032	63	PT	10295909	2020/03/03	TRUE
2022/02/14	20360	78	UK	10718565	2020/02/23	TRUE
2021/08/12	223	2	PL	37958138	2020/03/07	FALSE
2020/06/11	22	0	AT	8901064	2020/02/26	FALSE

Table 3.1: Illustration dataset, labeled by $high\_risk\_2w$

current_date	year	season	nr_months	ratio	cases_100k	current_risk	sum_2w	sum_2w_100k
2021/02/23	2021	winter	11	16.4	10.023	FALSE	33692	327.237
2022/02/14	2022	winter	24	261.0	189.951	TRUE	284573	2654.954
2021/08/12	2021	summer	17	111.5	0.587	FALSE	2453	6.462
2020/06/11	2020	spring	3	null	0.247	FALSE	471	5.292

Table 3.2: Generated variables, indexed by current_date.

Table 3.2 summarizes the variables generated by our algorithm when applied to the data in $\mathcal{D}$, shown in the previous table, and using the ER diagram in fig. 3.1. First, we find *year* resulting from a `decomposition` operation, computed by extracting the year from the *current_date* variable. Similarly, we have *season* that `maps` the *current_date* to the yearly season. `Algebraic` operations are illustrated through *ratio*, that is computed by dividing the number of *deaths* and *cases*, *nr_months*, which is the difference between *first_date* and *current_date* in months and *cases_100k* which is the number of *cases* divided by the *population*, multiplied by 100000. All of these variables are computed using only *record-based* operations, since the value of the new variable for a specific record only depends on the values of the inputs of that same record.

On the other hand, *sum_2w* is an example of a variable resulting from an `aggregation` operation, resulting from the sum of the number of *cases* from the last two weeks for the *country* under analysis (the country of the record whose value is being filled). Here, the *groupby* parameter defined in the relationship is the *country* variable, using for calculation the rows that have the same country and that the date is not larger than 15 days prior the *current_date*.

Finally, *sum_2w_100k* and *current_risk* could be seen as `composition` if we had omitted the *sum_2w* and *cases_per_100k*, respectively. Actually, they are just a division by the population and a comparison to a threshold (120 cases per 100k), after computing those previous variables.

So far, we have clearly defined the problem we are trying to solve and the operations that the algorithm is able to perform to solve it, illustrated by an example. While the purpose of the algorithm has been described, it was only done in a black-box manner. With everything defined, we can now go more in-depth to the various iterations of the algorithm and explain how it works, as well as the benefits and trade-offs of each iteration.

3.2 DANKFE-I algorithm

The DomAiN Knowledge based Feature Engineering (DANKFE) algorithm transforms the relationships between entities into new variables, when presented with an ER diagram and a dataset. The algorithm is presented in various versions that trade-off speed with versatility of operations that it the version is able to deal with.

The first version of the algorithm, *DANKFE-I*, is described in algorithm 1 and works as follows: the relationships are read from the ER model, and stored as a queue to be processed. The relationships are processed one by one, if the input variables for them are already available. If part of the input is not yet available (meaning that at least one of the input variables is not originally in the dataset and still in the queue to be processed), that relationship is sent to the end of the queue. If all the inputs are already available (have all been generated or are originally present in the input dataset), the list of operations specified in the diagram for that relationship is applied to any row in the dataset that satisfies the constraints imposed for the relationship. Whenever any row does not meet the constraints, a null value is imputed. When all rows are processed, the relationship is removed from the queue. The algorithm ends when the processing queue is empty.

Definition 5. *Given a dataset $\mathcal{D}$ described by a set of d variables, $\mathcal{F} = \{v_1, ..., v_d\}$ and an ER diagram, $\mathcal{KB} = (\mathcal{E}, \mathcal{R})$ as defined before, where for each $v \in \mathcal{F}$ exists an $e \in \mathcal{E}$ such as e represents v, the algorithm generates a new variable v' for each relationship $r \in \mathcal{R}$, extending the set of features $\mathcal{F}$ to $\mathcal{F}'$, and the original dataset $\mathcal{D}$ to $\mathcal{D}'$, by filling the new variables for all records in $\mathcal{D}$, according to the procedures described in definition 3.*

This version of the algorithm abides by definition 3, meaning it can perform *record-based* operations, since it processes the dataset one record at a time. If a relationship has multiple operations to be performed (composition operation), the algorithm applies each operation in the list of operations in reverse (similarly to a composition of operations) sequentially over the corresponding inputs (the values of the input variables defined in the relationship), returning the output value (assigned to the given record), processing the defined operations row by row.

The operations defined in section 3.1.1 are defined similarly to a Production Rule System (PRS), an if-then system which interprets the operation needed to generate a new value for a relationship, triggering the necessary action with the values of the inputs given to the algorithm from that relationship. For example, if the algorithm is processing a relationship found in the ER diagram that wants to create the *year* variable, where the input is *current_date* and the output is *year*, the algorithm's PRS senses if the operation defined in the relationship is the operation used to extract the year from the input from the list of operations, triggering the action that completes that calculation and returns only the year from the date, which is then written as the new value for the record being processed for the new variable. This

Algorithm 1 DANKFE-I algorithm

```
procedure DANKFE-I(𝒟, ℱ, 𝒦ℬ)
    queue ← 𝒦ℬ['relations']
    while queue is not empty do
        current_relation ← pop(queue)
        inputs ← current_relation['inputs']
        constraint ← get_constraint(current_relation['constraint'])
        operations ← reverse(current_relation['operations'])
        if inputs ∈ ℱ then
            args ← 𝒟[inputs]
            for operation in operations do
                for row in args do
                    if satisfies(row, constraint) then
                        row ← operation(args)
                    else
                        row ← null
                    end if
                end for
            end for
        else
            queue ← append(queue)
        end if
    end while
end procedure
```

process is then repeated for all rows that pass the possible constraint defined in the relationship, and in the end, the *year* variable is complete and part of the dataset.

Since there is no row dependence in *record-based* operations, meaning that these operations only require values of the row being processed by the algorithm, it can be done very efficiently using the Pandas function `apply` and using lambda functions in Python.

3.3 DANKFE-II algorithm

The second iteration of DANKFE extends the previous version by allowing the user to define *aggregation-based* operations, as described in definition 4. The algorithm is defined in algorithm 2.

Definition 6. *Given a dataset $\mathcal{D}$ described by a set of d variables, $\mathcal{F} = \{v_1, ..., v_d\}$ and an ER diagram, $\mathcal{KB} = (\mathcal{E}, \mathcal{R})$ as defined before, where for each $v \in \mathcal{F}$ exists an $e \in \mathcal{E}$ such as e represents v, the algorithm generates a new variable v' for each relationship $r \in \mathcal{R}$, extending the set of features $\mathcal{F}$ to $\mathcal{F}'$, and the original dataset $\mathcal{D}$ to $\mathcal{D}'$, by filling the new variables for all records in $\mathcal{D}$, according to the procedures described in definition 3 and definition 4.*

DANKFE-II works as follows: the relations are read from the ER diagram, and stored as a queue to be processed. The relations are processed one by one, if the inputs are already available, otherwise

Algorithm 2 DANKFE-II algorithm

```
procedure DANKFE-II(𝒟, ℱ, 𝒦ℬ)
    queue ← 𝒦ℬ.relations
    while queue is not empty do
        rel ← pop(queue)
        if rel.inputs ⊄ ℱ then
            queue ← append(rel)
        else
            for row in 𝒟 do
                if satisfies(row, constraint) then
                    if rel.groupby exists then
                        row′ ← get_rows(row, 𝒟[rel.inputs], rel.groupby, rel.condition)
                    else
                        row′ ← row
                    end if
                    for operation in reverse(rel.operations) do
                        row′ ← operation(row′)
                    end for
                else
                    row′ ← null
                end if
                𝒟′ ← 𝒟′ ∪ {row + row′}
            end for
        end if
    end while
    return 𝒟′
end procedure
```

the relationship is sent to the end of the queue. If the inputs are already available, the algorithm is ready to fill the new variable for each record in the original dataset. If the relationship requires an aggregation operation, i.e., the relationship specifies a *groupby* parameter, then we need to collect the rows that match the specified condition to perform the aggregation. Only after collecting the rows it is then possible to apply the list of operations which create the new values for the variable, for all the records satisfying the constraints imposed. The operations are applied as a composition of functions, beginning with the last one and sequentially applying the following ones. Whenever any row does not meet the constraints, a null value is imputed. When all rows are processed, the relationship is removed from the queue.

Similar to the first iteration of the algorithm, the aggregation operations are defined in a similar manner to the other operations. The algorithm's PRS interprets the operation needed to generate that value, but for these kinds of operations it also has access to the collected rows, triggering the aggregation operation with the input values of those rows, outputting the calculated value to the new variable of that record. For example, if the algorithm is processing a relationship found in the ER diagram that wants to create the *avg_temperature_week_per_city* , where the input is *current_date* and *temperature*, grouped by *city*, the algorithm collects the rows with a similar city to the record it is processing, but only the ones

whose date is within one week of the record. It then checks that the operation is the average, which triggers the calculation. The result (the average temperature for one week in that city) is then written as the new value for that record. The process is then repeated for all records that pass the possible constraint defined in the relationship, and in the end, the *avg_temperature_week_per_city* variable is complete and part of the dataset.

As stated before, this version of the algorithm is an extension of DANKFE-I, meaning it performs *record-based* operations in a similar fashion, using lambda functions in Python. This version of the DANKFE algorithm sacrifices some processing speed (since the process of collecting rows for generating variables that require *aggregation-based* operations takes longer time to run, which will be evaluated in section 4.3.2), but it adds the ability of performing these kinds of operations, which can greatly benefit the datasets depending on the available domain knowledge. Knowing useful aggregations for specific datasets can generate much useful information and improve the quality of machine learning models.

3.4 DANKFE-III system

As seen before, feature engineering (and subsequently, feature generation), is only a part of a pipeline of operations that turn raw data into possibly important information, known as the KDD or data science process. AutoML frameworks that automate this pipeline are currently not spending much time or resources into augmenting datasets, even less with the use of domain knowledge, but rather spending more computation in other parts such as data preprocessing, model selection and hyper-parameter optimization.

Similarly to how these frameworks work, to be able to reach the most robust models, other parts of the KDD process can be coupled with the DANKFE algorithm to yield the best results. The algorithm can be encapsulated into a new function that deals with data preparation before and after generating the features, before running machine learning models, which then can be further selected and optimized as well, but the scope of this work is focused on feature generation and the data preparation step. In chapter 4, we explain the process of evaluation of the extended datasets, where some model selection and hyper-parameter optimization techniques were also implemented to create a better comparison to the AutoML framework.

Additionally, not all variables require domain knowledge to be generated. Simple record-based operations such as decomposition of dates or aggregation-based operations such as a descriptive statistic of a numeric variable (mean, median, maximum, minimum, etc) can be easily generated by checking in which existing variables we are able to apply these operations (easily accomplished by checking their *type* in the ER model) and by doing so, adding these new relationships to the ER model, thus creating new variables. For this to be possible, there needs to be a template where the relationships are

described, with the necessary operations for creating this variables, where only the *input*, *output* and *groupby* parameters need to change based on the data. An example of a variable template is in fig. 3.2 (left). This template relationship is similar to the relationships found in the ER models, but with those three parameters missing.

The created pipeline can be seen in fig. 3.3. We can see that data preprocessing techniques can be applied before the generation of features with operations such as missing value imputation, dummification/discretization, label encoding, etc. Afterwards, automatic variables can be added to the ER model if required by the user. Then, the DANKFE algorithm runs (either the first or second version depending on whether the ER model has relationships that create variables which require *aggregation-based* operations), and finally additional processing can be applied to the data, such as scaling or balancing of variables, if chosen by the user. These two operations need to be done after feature generation to ensure the generated variables keep a similar behavior with the rest of the data and are correct when generated.

To implement this, a configuration JSON file is required at the beginning, as well as the dataset and ER model, where the preparation techniques can be programmed per user choice. The configuration file can be seen in fig. 3.2 (right). The user can specify whether the pipeline checks missing values (before the algorithm is run), scaling, balancing (both after the variables are generated) and if augments the ER model by automatically decomposing dates and/or generating the aggregated summary for numeric variables (max, min, average, standard deviation and median), given another variable to perform the aggregation (the *groupby* parameter). The configuration file changes the parameters in the template relationships and introduces these new relationships inside the ER model, so that DANKFE can generate them. Therefore, if these automatic features are generated by the user, they are added to the domain knowledge in the model, and DANKFE generates them in the same manner as if they were in the model right from the start. From the example in fig. 3.2 (right), the input dataset would be cleaned and checked for missing values, dates would be decomposed automatically and a total of 10 aggregation-based variables would be added (5 for the summary of *cases* per *country* and another 5 for the summary of *deaths* per *country*). Scaling and balancing would be performed after the features are generated.

The configuration file allows the user to select some techniques for each data preparation step, with the ability of choosing which one is more suitable for the task. Missing values can be imputed using the median for numeric variables and the mode for symbolic variables. A label encoder can also be used to turn symbolic variables into ordinal ones. Scaling can be done either by *z-score* or *minmax*, and balancing is done via an hybrid approach that depends on the proportion of the data. If the proportion between the positive and negative class is above two thirds (0.66%), then it balances the records either by oversampling the class with less records, or if the number of records for any class is above 25000, it balances them into this amount (either by undersampling both classes, or oversampling one and

```
"dates":
[{
  "name": "day",
  "type": "int",
  "operations": ["getDay"],
  "inputs": [],
  "output": "day",
  "groupby": "",
  "needsRows" : 0,
  "constraint": ""
},
```

```
{
 "checkMissingValues" : "auto",
 "checkScaling" : "zscore",
 "checkBalancing" : true,
 "generateDates" : true,
 "generateFiveSummary" : ["cases","deaths"],
 "groupby": ["country"]
}
```

Figure 3.2: Example of automatic variable template (left) and configuration file (right).

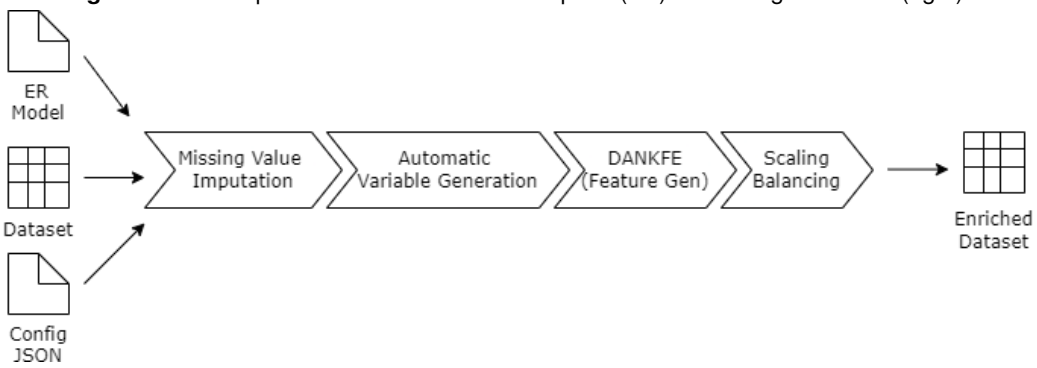

Figure 3.3: Data preparation and feature generation pipeline.

undersampling another). This allows for the models to keep reaching strong results but with a lower number of records in a more distributed fashion, which helps in some machine learning models.

While the DANKFE-III system does not change how the features are generated, it helps automate the entire data preparation step, which can be very beneficial for the later data mining stage. This way, some features can still be generated even with very little domain knowledge (only knowing which variables are dates or numerical), which still augments the feature space and possibly improve the amount of information that ML models can extract. It also facilitates data cleaning and preprocessing, ensuring that the enhanced datasets are ready for data mining. As we will see in chapter 4, these data preparation steps are very important in the improvement of the performance of models, but a large impact of that improvement is still through the use of domain knowledge.

4

Case Studies

Contents

In this chapter, we proceed with the evaluation of our proposed solution, explaining the case study where the DANKFE algorithms were tested and giving information regarding the datasets used in the case study, the evaluation methodology that was followed for the evaluation of the algorithm, as well as other statistics.

In order to validate our proposal, we will analyze the performance of several machine learning models trained over a case study composed of several datasets, which will be described in detail in section 4.1. More information regarding the evaluation methodology is present in section 4.2 and the results are shown in section 4.3 for each version of the DANKFE algorithm.

4.1 Datasets Description

In order to discern whether the use of domain knowledge inserted into data can improve the amount of information that can be extracted from it, a case study needed to be developed where, as part of the KDD process (explained in section 2.1.1), after the process of cleaning and processing a dataset to maximize the possible useful information that can be extracted, data mining can be performed. The solution was to develop a case study composed of several datasets with different domains, so that our approach also proves to be domain-agnostic (it works for all domains as long as knowledge on that domain is present). The datasets collected are available in table 4.1.

Dataset	URL
COVID-AF	https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide (only African countries)
COVID-AM	https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide (only American countries)
COVID-AS	https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide (only Asian countries)
COVID-EU	https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide (only European countries)
COVID-OC	https://www.ecdc.europa.eu/en/publications-data/download-todays-data-geographic-distribution-covid-19-cases-worldwide (only Oceanian countries)
CRIME	https://data.world/data-society/city-of-baltimore-crime-data
AQ	http://tapdata.org.cn/?page_id=931
Energy	https://data.world/makeovermonday/2019w32
GCCD	https://data.world/data-society/global-climate-change-data

Table 4.1: Datasets under analysis.

These datasets allowed for the generation of variables with different natures, as long as domain knowledge is provided. To solve this, for each dataset an ER diagram and a relevant binary class variable was created. Samples of the ER diagrams used for the generation of variables are presented in this section. Having understood the domains and goals of the process we are evaluating (step 1), and with the datasets collected and preprocessed (steps 2 and 3), we followed the KDD process by matching our goal to a classification problem. Step 4 of the KDD process is feature engineering, which is our subject of study with the DANKFE algorithm, and whose results we will show below.

Continuing to follow the KDD process, an EDA was done (step 6), to better understand the underlying behavior of the data, characterizing it via its granularity, distribution, sparsity and dimensionality. Since the class distribution of each dataset is also important to figure out the best metric to evaluate the machine learning models, this was also studied. What follows is a short description of each dataset,

listing its dimensionality (size in terms of rows and variables), as well as any correlations between the baseline datasets and the meaning of the class variable. A summary of this information, as well as the class distribution (balancing representing the proportion between the positive and negative class) is present in table 4.2, as well as additional profiling information on each dataset, such as boxplots and correlation graphs before and after the generation of variables in this section.

4.1.1 COVID Dataset

The COVID dataset was split into five different datasets, one for each continent, for more sound results. All except Europe are unbalanced and feature between 13 thousand and 23 thousand records, except for the Oceania dataset which features 2 thousand. It has 7 variables (8 for Europe), 3 of which are numeric. The class variable generated was `high_risk_2k`, which is True if the current day is a high risk day depending on the sum of cases for the last 15 days. It is also interesting to note that `cases` and `deaths` are highly correlated in Africa, America and Asia. The ER diagram in fig. 4.1 is a sample of the ER used for all datasets, except for the Europe dataset which has an additional feature generated, *holiday*, which is true if the current date is an holiday.

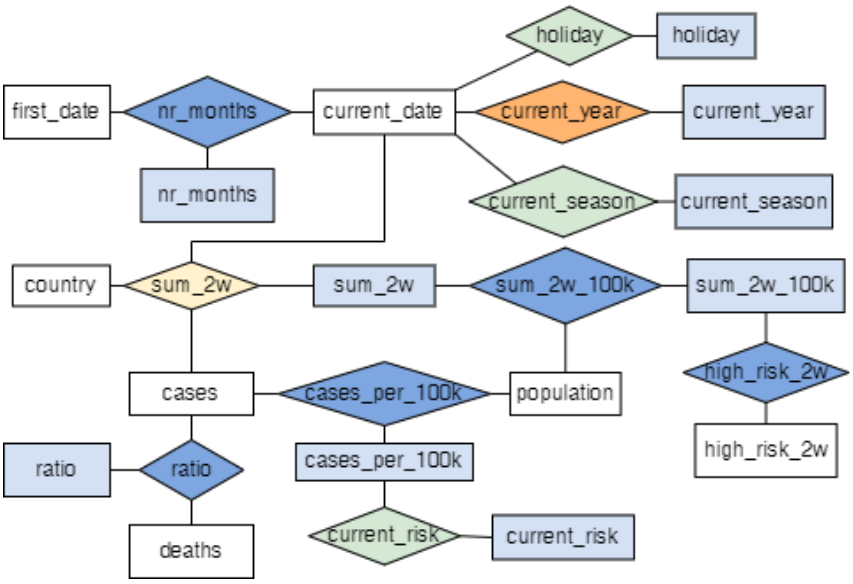

Figure 4.1: ER Diagram for COVID-Based datasets.

4.1.1.A Africa

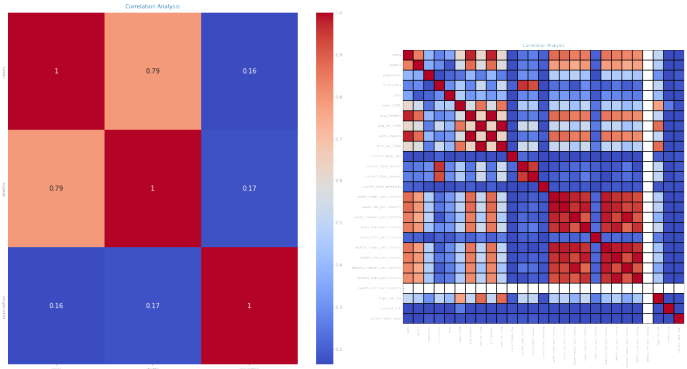

Figure 4.2: Correlation analysis before (left) and after (right) generation.

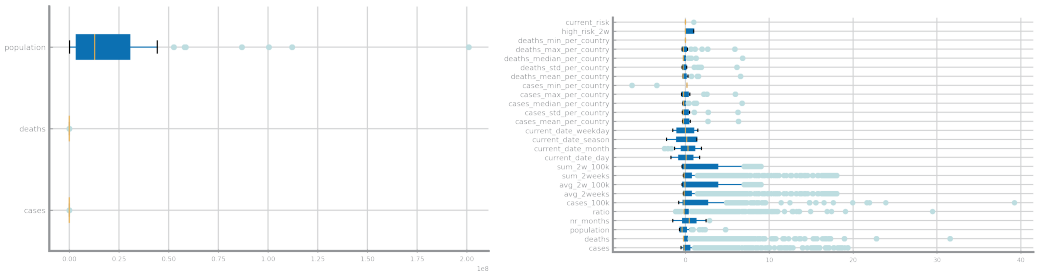

Figure 4.3: Boxplots for variables before (left) and after (right) generation.

4.1.1.B America

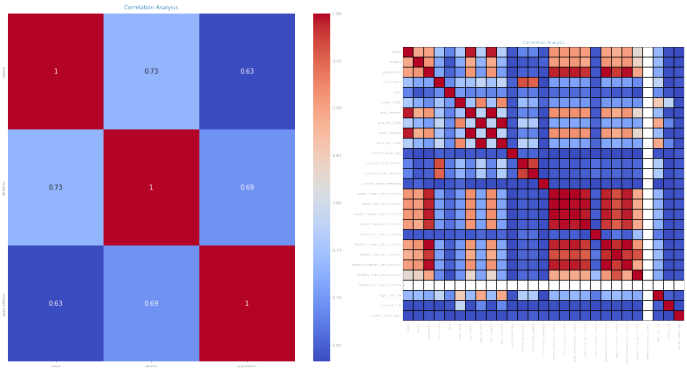

Figure 4.4: Correlation analysis before (left) and after (right) generation.

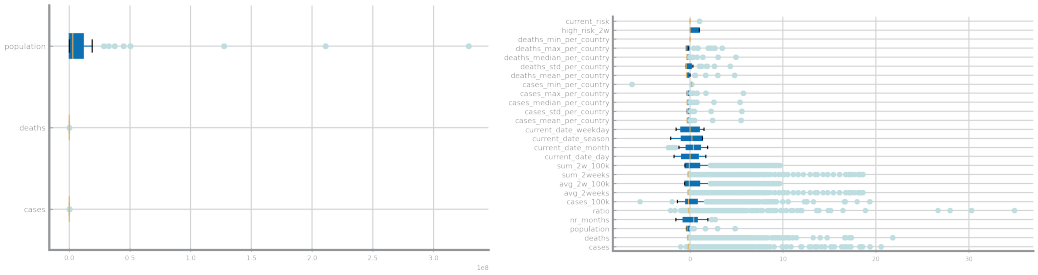

Figure 4.5: Boxplots for variables before (left) and after (right) generation.

4.1.1.C Asia

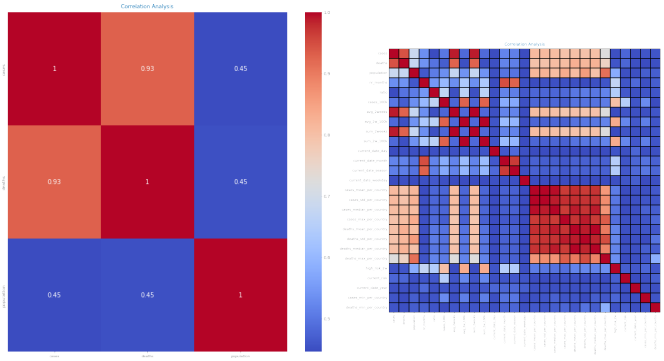

Figure 4.6: Correlation analysis before (left) and after (right) generation.

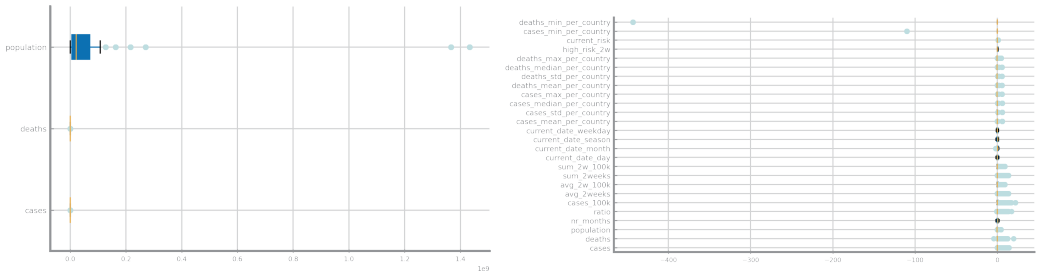

Figure 4.7: Boxplots for variables before (left) and after (right) generation.

4.1.1.D Europe

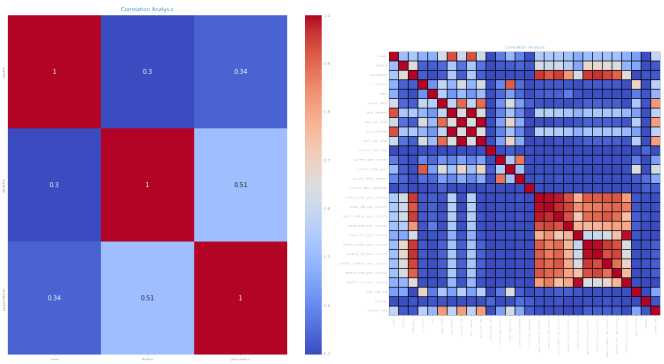

Figure 4.8: Correlation analysis before (left) and after (right) generation.

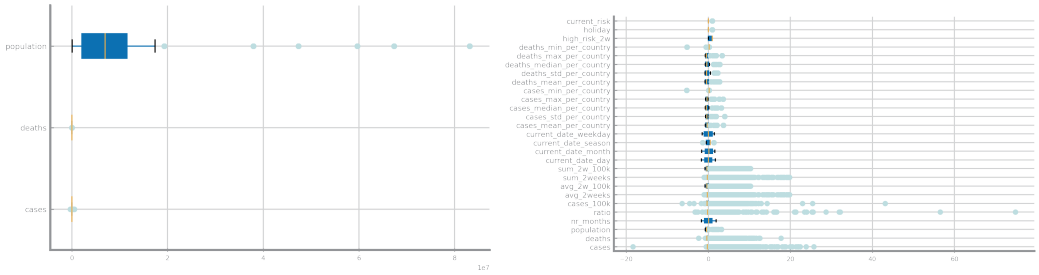

Figure 4.9: Boxplots for variables before (left) and after (right) generation.

4.1.1.E Oceania

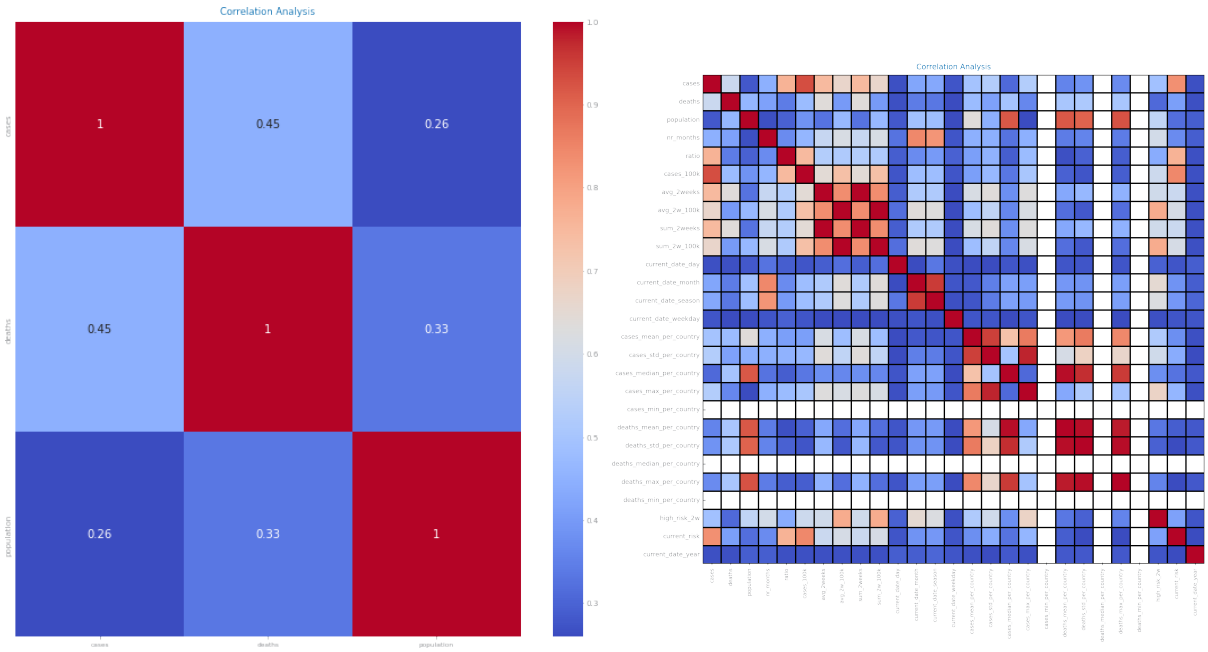

Figure 4.10: Correlation analysis before (left) and after (right) generation.

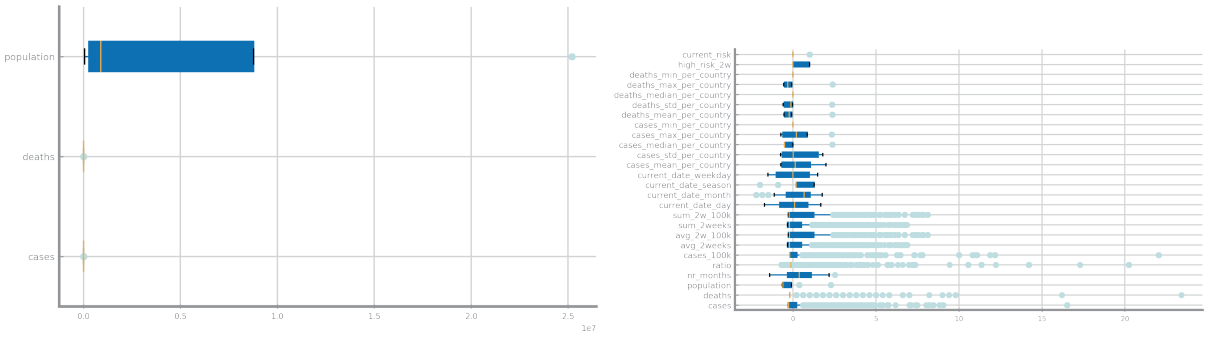

Figure 4.11: Boxplots for variables before (left) and after (right) generation.

4.1.2 AQ Dataset

The AQ (Air Quality) dataset reports various quantities of gases and other pollutants in cities in China. It has 94194 records, 29 variables (26 numeric) and it is also very unbalanced. Each pollutant has a column for its mean, maximum, minimum and standard deviation for each day, and some of these columns have strong correlations (such as `PM2.5_min` and `PM2.5_max`, for example). The class variable is `alarm`, which signals if the air quality is considered unsafe for a specific city in a specific date.

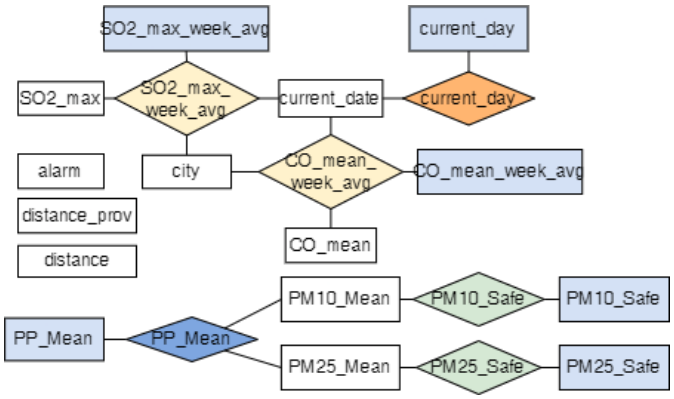

Figure 4.12: ER Diagram for the AQ dataset.

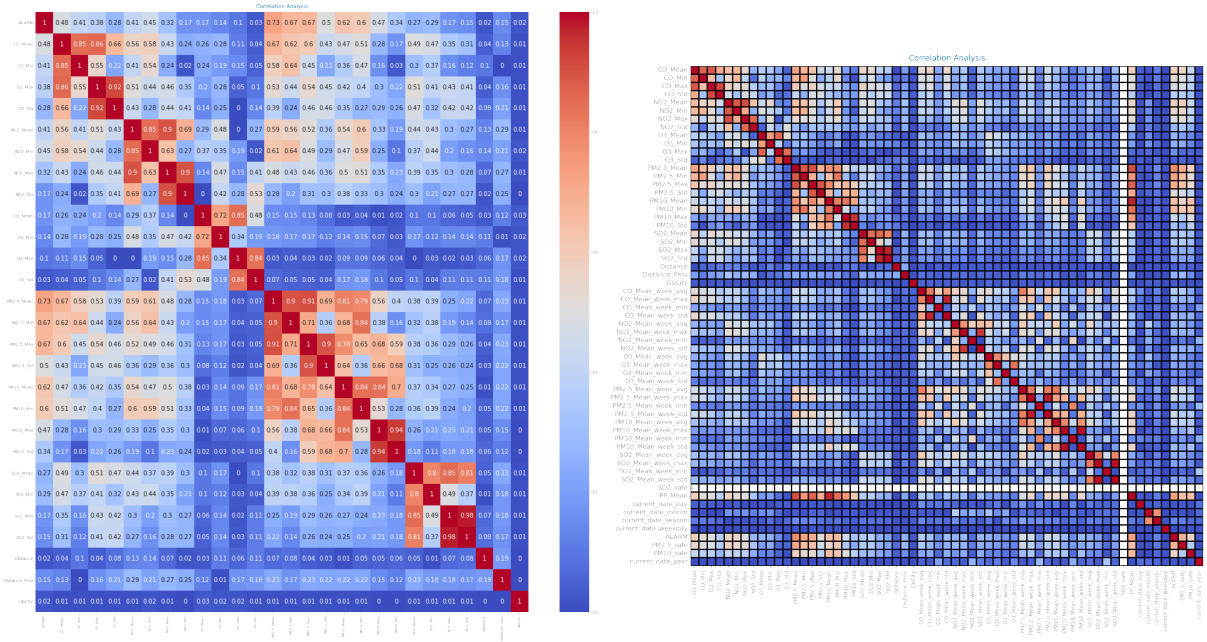

Figure 4.13: Correlation analysis before (left) and after (right) generation.

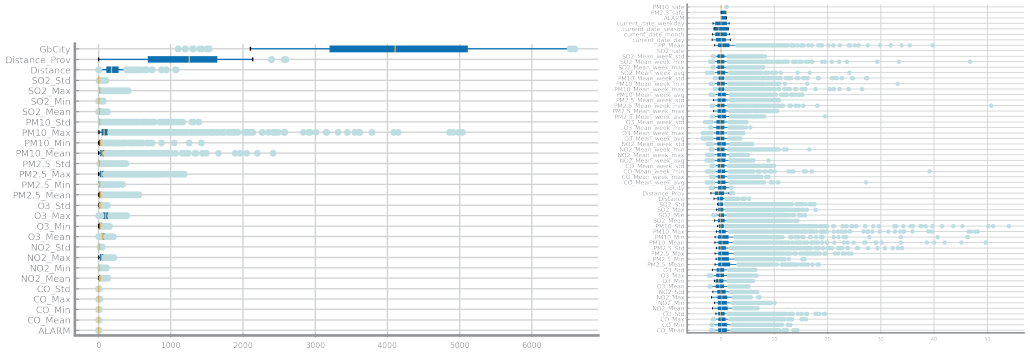

Figure 4.14: Boxplots for variables before (left) and after (right) generation.

4.1.3 Crime Dataset

The Crime dataset reports the occurrences of various crimes in the city of Baltimore and their description. It has 27994 records and 10 variables (3 numeric and 5 ordinal) and it is unbalanced. There are no strong correlations among variables in the baseline. The class `common_dist` is True if the crime happens frequently in that district.

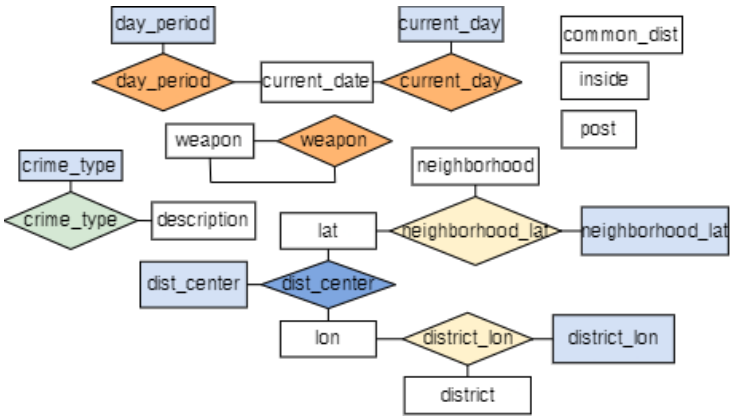

Figure 4.15: ER Diagram for the Crime dataset.

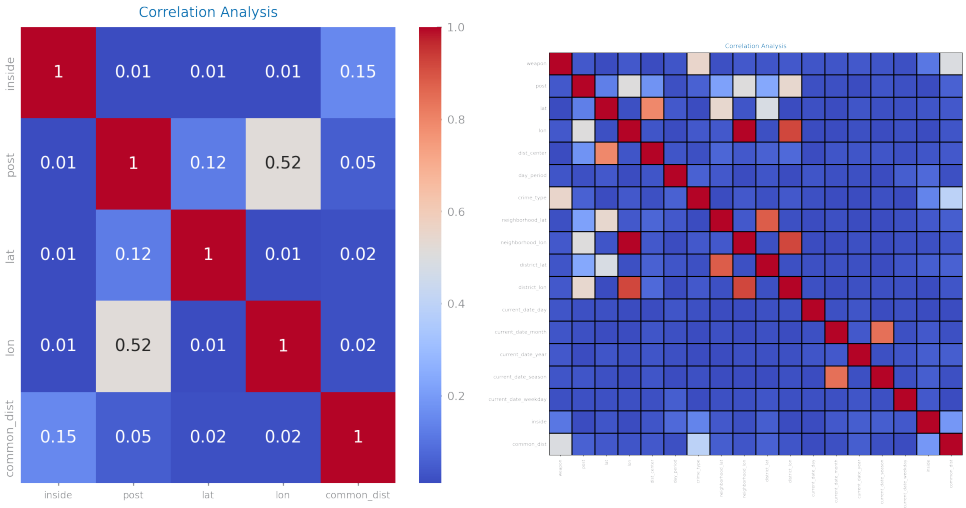

Figure 4.16: Correlation analysis before (left) and after (right) generation.

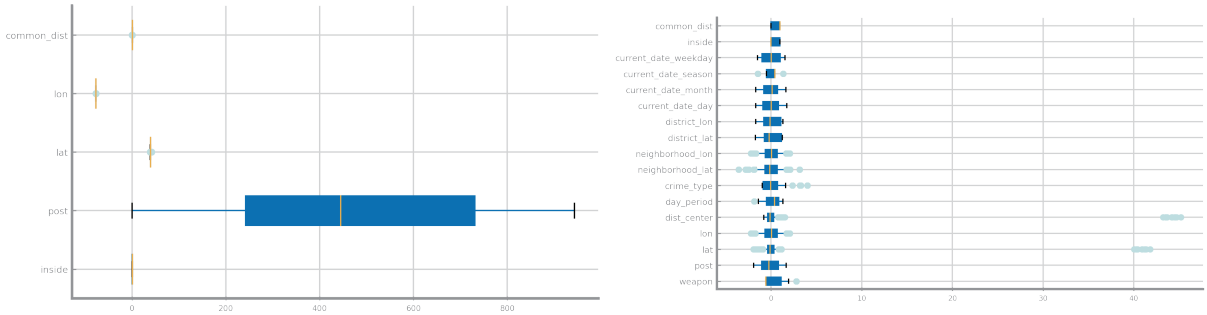

Figure 4.17: Boxplots for variables before (left) and after (right) generation.

4.1.4 Energy Dataset

The Energy dataset reports the amount of energy produced and spent in the United Kingdom every 5 minutes. It has 105408 records and 14 variables (12 numeric), and it is balanced. There are no strong correlations. the class `prod_vs_avg` is True if the energy production is above the hourly average.

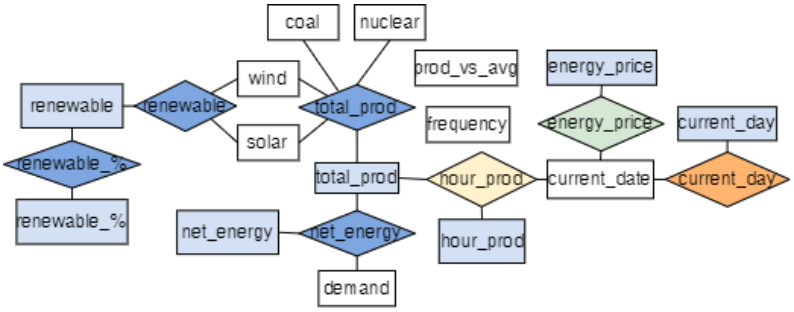

Figure 4.18: ER Diagram for the Energy dataset.

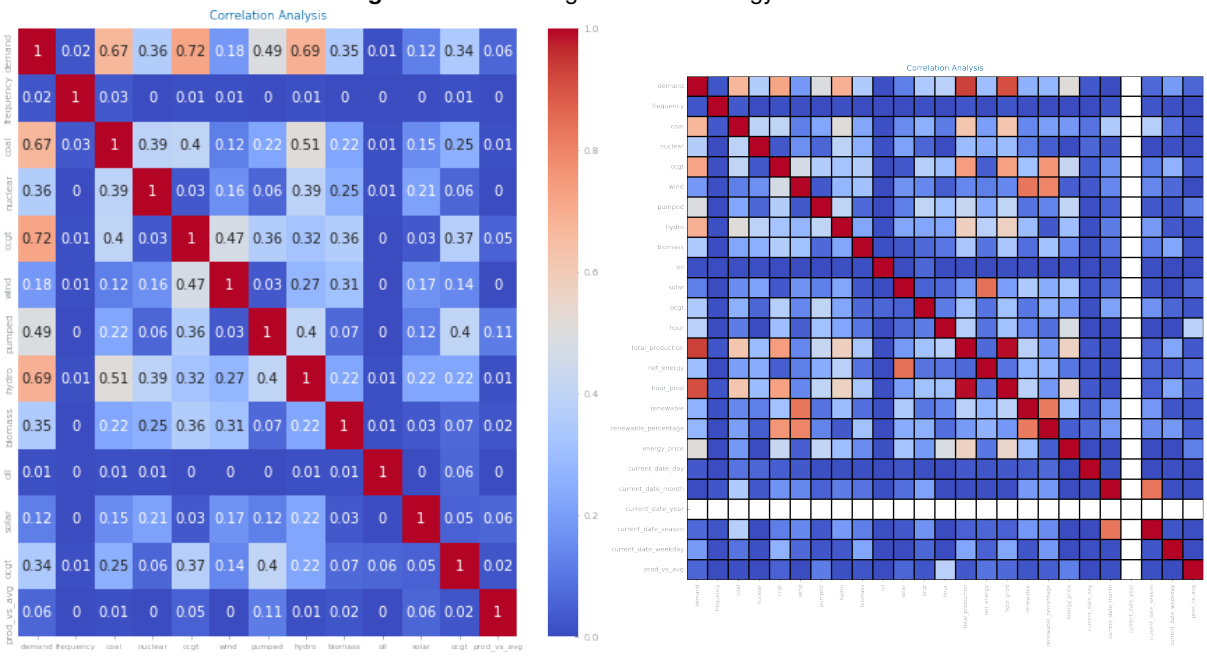

Figure 4.19: Correlation analysis before (left) and after (right) generation.

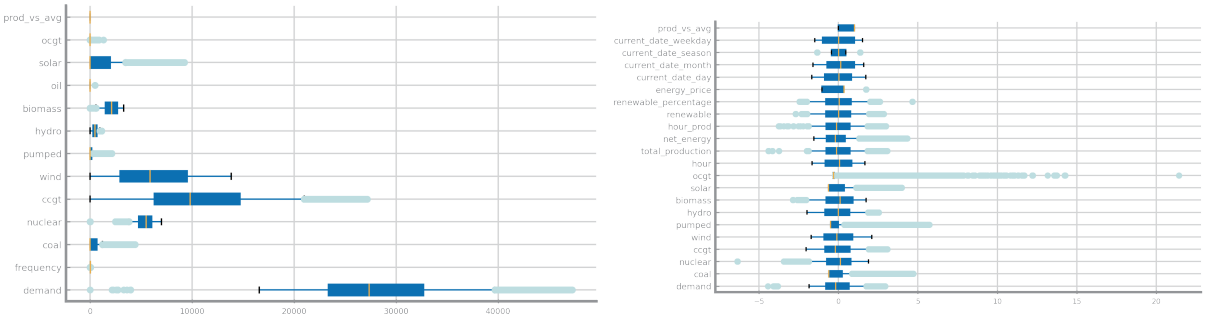

Figure 4.20: Boxplots for variables before (left) and after (right) generation.

4.1.5 GCCD Dataset

The GCCD (Global Climate Change Data) dataset reports the temperature for various cities around the world from 1900 to 2012, from the first day of each month. It has 32544 records, 8 variables (4 numeric and 2 ordinal) and is balanced, with no strong correlations between variables. The class `average_diff_pos` is True if the daily temperature is above the yearly average.

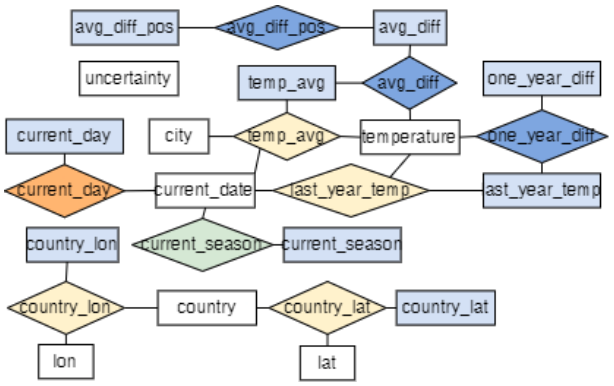

Figure 4.21: ER Diagram for the GCCD dataset.

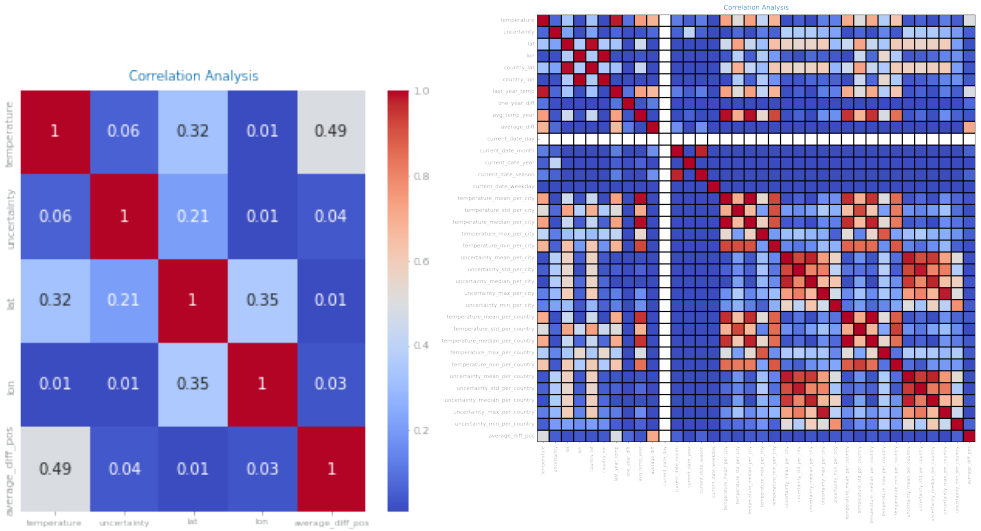

Figure 4.22: Correlation analysis before (left) and after (right) generation.

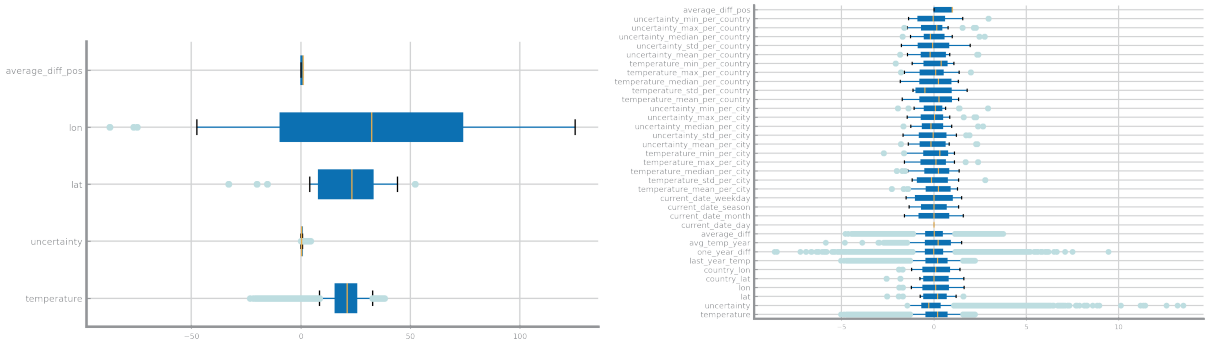

Figure 4.23: Boxplots for variables before (left) and after (right) generation.

	Baseline			DANKFE1			DANKFE2			DANKFE3		
Dataset	Rows	Variables	Balancing (%)	Rows	Vars	Bal (%)	Rows	Vars	Bal (%)	Rows	Vars	Bal (%)
Covid AF	14076	7	96.65	14076	11	96.65	14706	20	96.65	23029	30	58.47
Covid AM	13742	7	79.14	13742	11	79.14	13742	20	79.14	18360	30	56.21
Covid AS	13241	7	85.06	13241	11	85.06	13241	20	85.06	18716	30	57.05
Covid EU	22211	7	58.99	22211	12	58.99	22211	21	58.99	21701	31	58.99
Covid OC	2441	7	90.17	2441	11	90.17	2441	20	90.17	3560	30	57.72
AQ	94194	29	91.87	94194	33	91.61	94194	60	91.61	78259	62	65.03
Crime	27994	10	75.26	27994	13	75.26	27994	14	75.26	37895	19	55.6
Energy	105408	14	51.57	105408	19	51.57	105408	25	51.57	105408	26	51.57
GCCD	32544	8	52.36	32544	8	52.36	32544	22	52.36	32544	39	52.36

Table 4.2: Number of records, variables and class balance for the baseline and each extended dataset.

As seen in table 4.2, the number of variables increases for every version of the algorithm, since each version of DANKFE improves on the previous in terms of functionality, being able to generate more types of variables (aggregation-based variables are generated in DANKFE-II, and automatic date decomposition and summaries for numeric variables are generated in DANKFE-III).

It is also important to note that the number of rows changes for all datasets except Covid-EU, Energy and GCCD for the DANKFE-III version because balancing techniques were applied, as explained in section 3.4. Since all datasets were divided into training and testing samples (as will be explained in section 4.2), balancing was only applied to the training sample, since the test sample cannot be balanced as it will change the behavior of the data and bias results.

4.2 Evaluation Methodology

We had stated before our goal of figuring out whether feature generation through the use of domain knowledge can improve model performance in the KDD process, by creating a case study where the various datasets will be evaluated. Now, the details of the evaluation methodology need to be defined.

Models can be evaluated and compared in terms of their simplicity, their certainty or their utility, as referred in section 2.1.1. The importance given to any of these aspects depends from context to context and where and how the models will be applied. Here, certainty (model performance in regards to specific metrics) will be considered most important, and various machine learning models will be evaluated with varying degrees of simplicity.

To validate the three versions of the DANKFE algorithm, the baseline versions of each dataset, as well as the extended versions of each iteration of the algorithm were tested (for DANKFE-III, the final version of the dataset after processing was evaluated), both in terms of efficacy, with well-defined metrics that will be explained below, as well as efficiency, comparing the time performance of each algorithm. All approaches were compared with a popular AutoML framework (auto-sklearn [8]), in order to compare the strength of generating features with the use of domain knowledge, when compared to a framework that only uses simple cleaning and processing in terms of data preparation, giving a much larger focus in model selection and hyper-parameter optimization.

To understand the differences in the behavior of the algorithm when changing the dataset size, a scalability study was also performed.

To study the performance of models in regards to their efficacy, several evaluations metrics can be used, ones more suitable than others depending on the context and properties of the data. **Accuracy** is the number of correctly classified records divided by the total amount of records. We will use the terms TP for True Positives (records correctly classified as positive - will not survive), TN for True Negatives (records correctly classified as negative), FP for False Positives (negative records classified as positive) and FN for False Negatives (positive records classified as negative). Accuracy can be calculated as such:

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.1}$$

To spot possible strong or weak points in the models (for example, too many false positives or negatives), we can use other evaluation metrics. We will use metrics such as **precision**, which is the percentage of records classified as positive which are truly positive:

$$precision = \frac{TP}{TP + FP} \tag{4.2}$$

We will also use **recall**, also known as true positive rate *TP rate*, which is the percentage of positive records that the model correctly recognizes.

$$recall = \frac{TP}{TP + FN} \tag{4.3}$$

To balance precision and recall, metrics such as the **F-Score** are used. Depending on which of the two metrics we value most, we can assign weights to each one. Giving the same weight to both precision and recall, we have the **F1-Score**, which is their harmonic mean.

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)} \tag{4.4}$$

We can also use **specificity**, also known as true negative rate *TN rate*, to measure the amount of negative records that the model correctly recognizes.

$$specificity = \frac{TN}{TN + FP} \tag{4.5}$$

Recall and specificity are closely related. There is a trade-off between these two values until one of them reaches 100% (either having 0 false negatives or 0 false positives, respectively). Beyond this point on either side, both decrease. To better understand how they relate in a model, we can plot the **ROC curve**, which shows the balance between the rate of recognized positives and negatives. To create it,

we take the recall/TP Rate and the **FP Rate** ($1 - TNRate$), plotting one against the other with various thresholds. If the result corresponds to the diagonal line $y = x$, then our model is no better than a random classifier. The closer we get to the point (0,1), the higher our specificity and recall are, and therefore the better the classifier is. We can use the **AUC** (Area Under the Curve) to evaluate the model, as it is equivalent to the probability that the model will rank a random positive instance higher than a random negative instance [86].

As previously seen, most datasets are unbalanced. While all metrics will be taken into account when training and selecting the best model, AUC will be considered the leading metric.

Since the goal of the solution is to improve automatic machine learning frameworks with the use of domain knowledge and feature generation, the behavior of model evaluation was made similar to the process that AutoML frameworks run. To achieve more robust results, several training techniques (Naive Bayes, K-Nearest Neighbors (KNN), Decision Trees, Random Forests and Gradient Boosting, all implemented with the `scikit-learn` package) were used to find the 10 best models trained over an equal number of random data partitions (into training and testing sets).

For each machine learning model, the hyper-parameters were tuned using Grid Search, which explores every defined combination of hyper-parameters, returning the one with that yields the best result. Since this process can be very time-consuming, if a model achieved a strong enough result with a combination of hyper-parameters (over 98% accuracy and 90% F1-Score), then it would return that specific combination. The 10 optimized models for each dataset were then averaged.

4.3 Results

4.3.1 DANKFE-I Results

As stated in section 3.2, the DANKFE-I algorithm cannot exploit row dependencies, therefore it only works with record-based operations. The results obtained from the extended datasets are going to be compared to the baseline where no features were generated. Since DANKFE-I does not do any data preprocessing techniques, the baseline also did not receive any (except the imputation of a few missing values, which was necessary for the training of models).

Results in fig. 4.24 (left) show the average AUC for all datasets, for each machine learning model. We can see that feature generation guided by domain knowledge improved the performance over the baseline. The models that have benefited the most are KNN and Gradient Boosting, with an increase in AUC in almost 5 percentage points (pp). It is also interesting to see that auto-sklearn achieves a result very similar to the original dataset without feature generation, and KNN and Gradient Boosting achieve in average results above this framework, by generating variables that require record-based operations.

Fig. 4.24 (right) shows the average time per record spent by each machine learning technique,

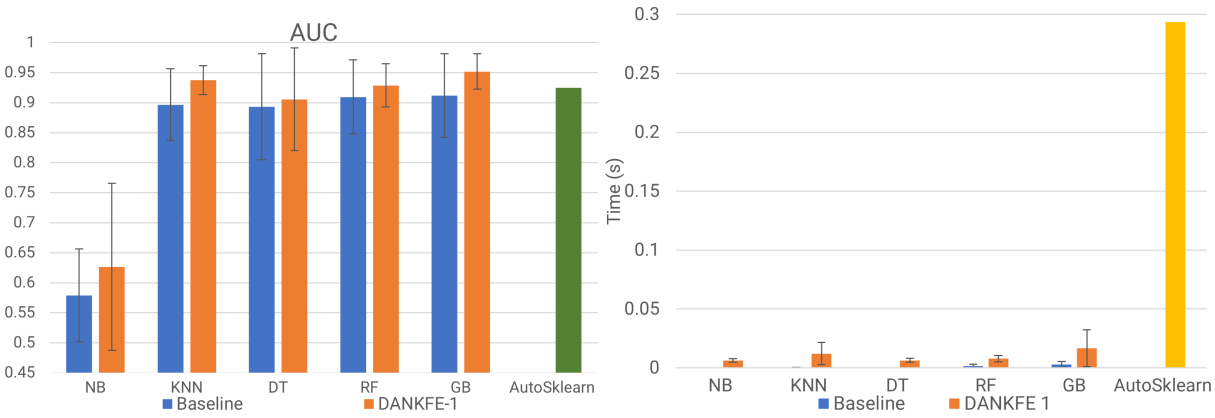

Figure 4.24: Quality of models (left) and processing times (right) for different machine learning algorithms.

already including the time spent running DANKFE-I for extending the dataset. We can see that the time spent in feature generation and model training has a small increase, which stays somewhat constant independently of the model used, at around 10 to 20 milliseconds per record. The results become much better when compared to auto-sklearn, which takes a fixed time of one hour to run, corresponding to around 300 milliseconds per record.

The efficacy of the DANKFE algorithm can not only be measured by the model scores, but also by measuring he impact of the generated features in these models, which can tell us if the features we are generating are providing useful information. Fig. 4.25 shows the average feature importance for Decision Trees, Random Forests and Gradient Boosting. The generated features have made the most impact in COVID-based datasets, making up almost all the importance given by the algorithms, and they also benefited the other datasets, especially in Random Forests.

The type of variable generated also influences the time spent, as seen on fig. 4.26 (left). Since all record-based operations are performed using the *apply* function in the *Pandas* library, and they do not require any additional rows, their time is somewhat similar, only depending on the amount of operations performed per dataset. The distribution of operations per dataset can be seen on fig. 4.26 (right). Decomposition and mapping operations take similar time to run, with algebraic operations either taking less or more time than the other two depending on the dataset.

4.3.2 DANKFE-II Results

The DANKFE-II algorithm, which introduces aggregation-based operations, was evaluated in the same manner as the previous version. Since it also does not feature any data preprocessing techniques on the dataset (apart from simple missing value imputation to enable data mining processes), the baseline was also not preprocessed. The evaluation method was similar as before, with multiple machine learning algorithms trained 10 times on random data partitions with the same size, and grid search. The results

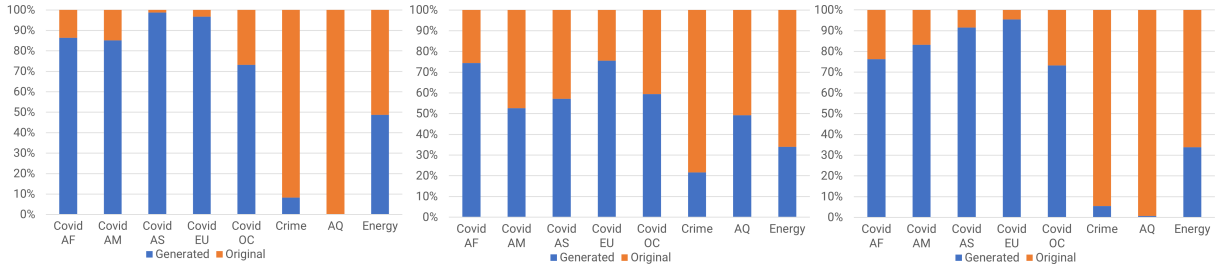

Figure 4.25: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

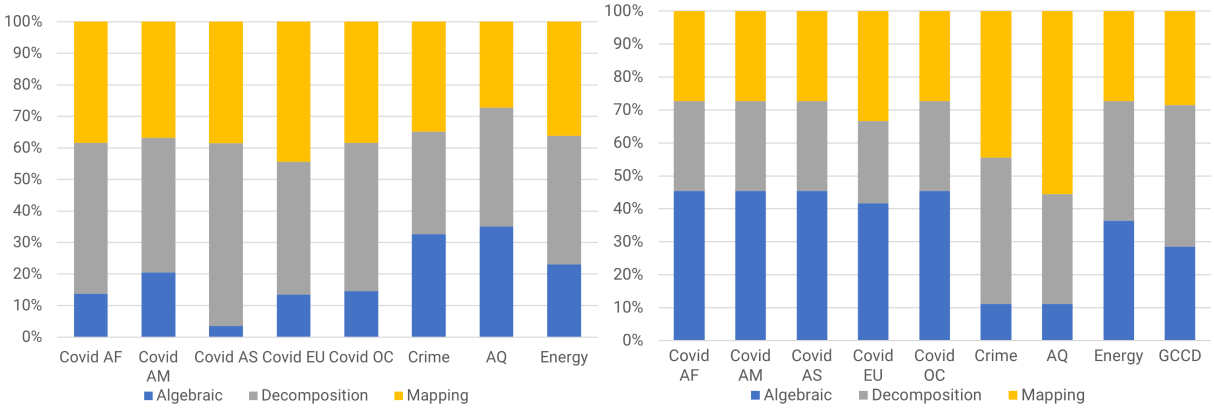

Figure 4.26: Time comparison (left) and amount of generated features (right) per type of operation.

from auto-sklearn are the same as in DANKFE-I, because they were trained in the baseline datasets.

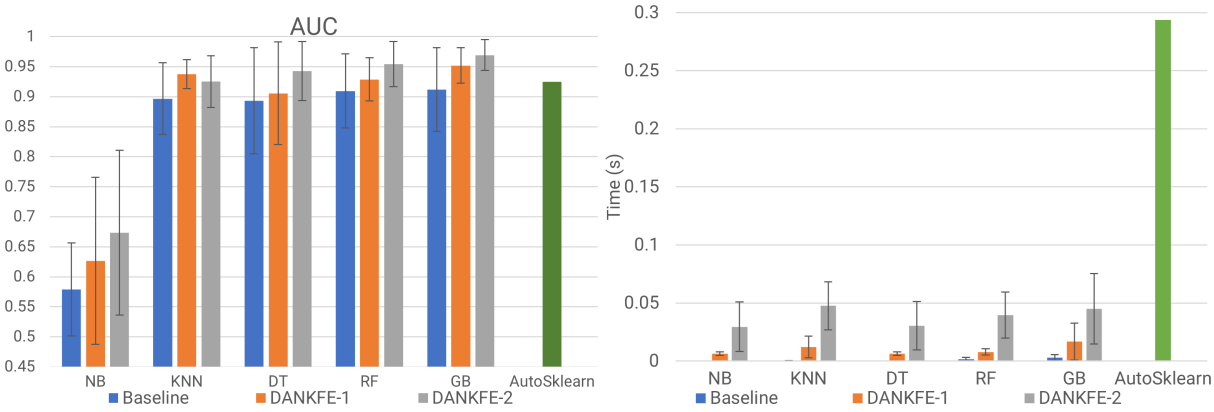

Figure 4.27: Quality of models (left) and processing times (right) for different machine learning algorithms.

As seen in fig. 4.27, the addition of aggregation-based operations leads to an overall increase in results for all algorithms except KNN. Naive Bayes and Decision Trees benefited the most, with an improval of around 5 pp. On average, model results with DANKFE-II surpass auto-sklearn, with a maximum difference of 4.6 pp. for Gradient Boosting.

In terms of time spent, features that require aggregation-based operations can take much longer to generate compared to ones that do not need them, due to the fact that they require rows to be collected

and aggregated (with the `groupby` parameter in the ER diagram for that variable), which can take a longer time to run. Nevertheless, the time spent per record still remains somewhat constant independently of the algorithm used, taking approximately 15% of the time spent by auto-sklearn for the entire process (generation + model selection and optimization).

Aggregation-based operations have also caused a big impact in feature importance for the machine learning models. Fig. 4.28 shows that apart from the Crime dataset, the generated features were given the most importance by far, in Decision Trees, Random Forests and Gradient Boosting. The largest difference compared to the previous version of the algorithm can be seen in the AQ dataset, where aggregation-based operations became the most important for the models.

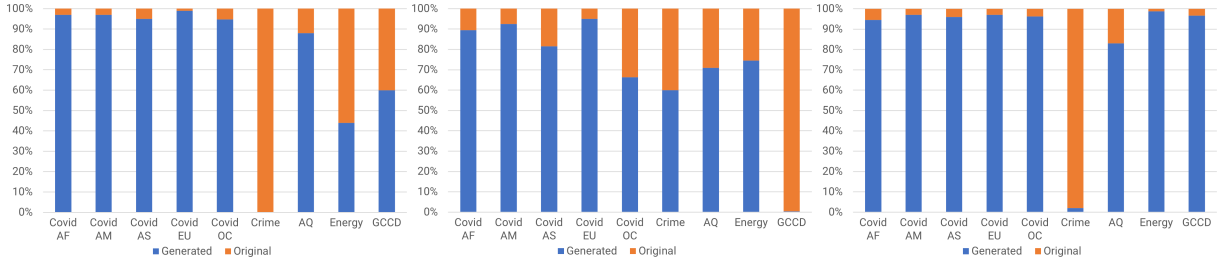

Figure 4.28: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

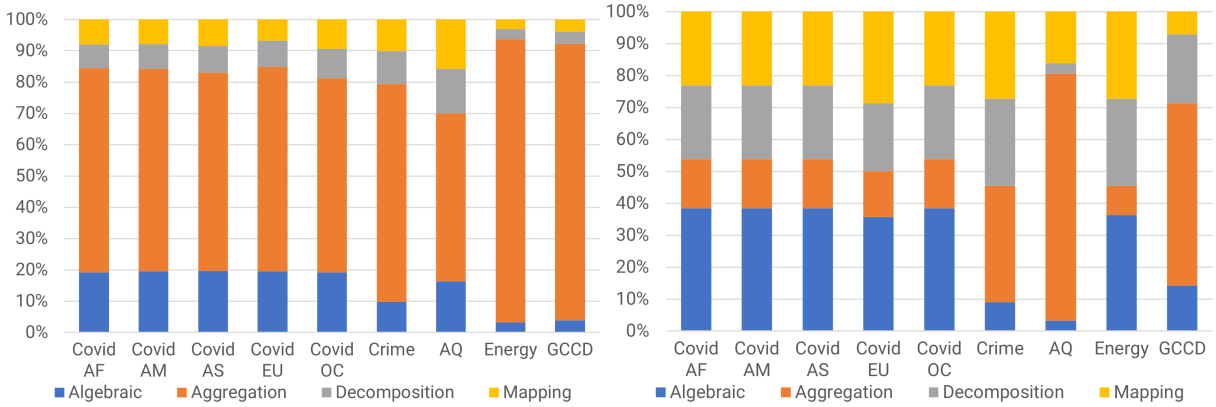

Figure 4.29: Time comparison (left) and amount of generated features (right) per type of operation.

As stated before, aggregation-based operations take longer to run, depending on the amount of records needed for aggregation before the value for the new variable is calculated. Fig. 4.29 (left) confirms this. Even though only the AQ and GCCD datasets have the majority of generated features needing aggregation-based operations, as seen on fig. 4.29 (right), this type of operation is the one that takes longer to run.

4.3.3 DANKFE-III Results

DANKFE-III couples the feature generation algorithm inside a data preparation pipeline, which ensures the datasets not only benefit from being enriched with the use of domain knowledge, but also improve on possible pitfalls that can appear while data mining, such as being unbalanced, having data with different scales, having missing values which require imputation, nominal data which requires encoding, etc. Some of these issues can decrease the performance of machine learning models, while others can even prevent them from running and need immediate fixing.

For fairer comparison, the baseline datasets were also preprocessed, using the same scaling and balancing techniques in order to study the impact of the feature generation algorithm alone.

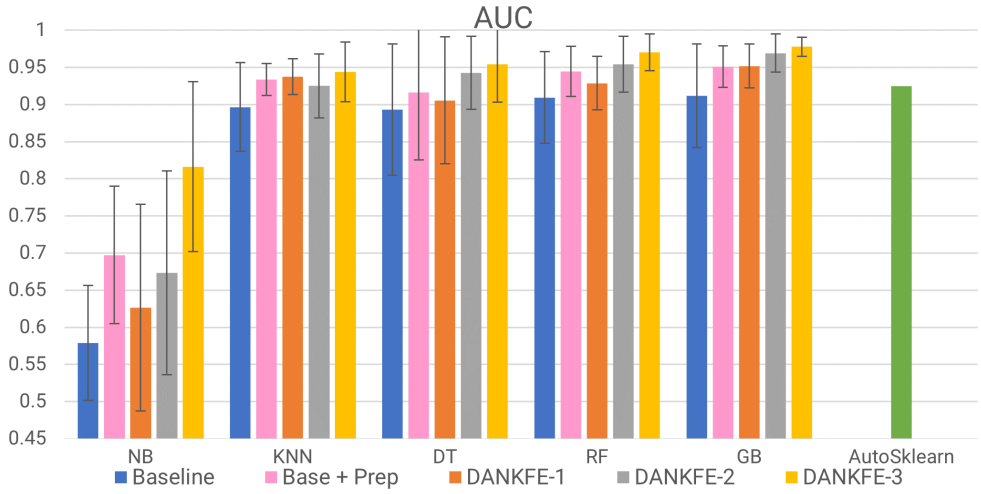

Figure 4.30: Quality of models (left) and processing times (right) for different machine learning algorithms.

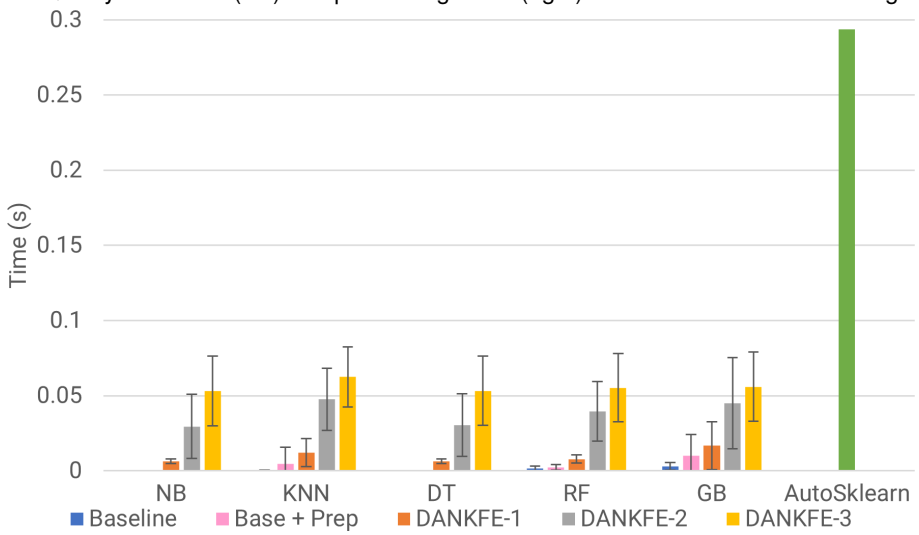

Figure 4.31: Quality of models (left) and processing times (right) for different machine learning algorithms.

Fig. 4.30 shows that adding data preprocessing techniques can greatly increase the performance of

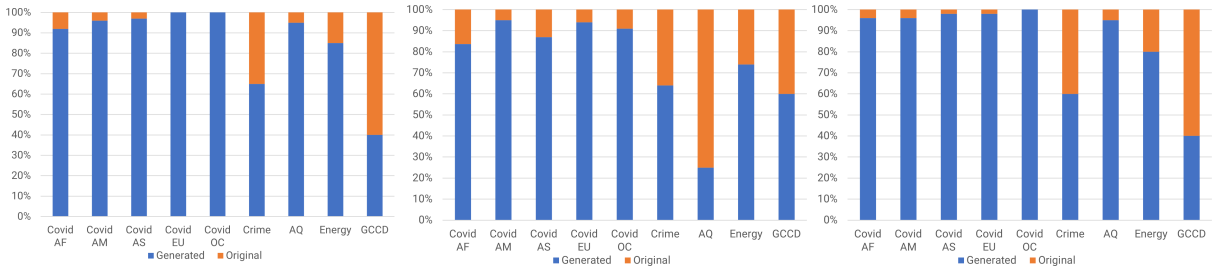

Figure 4.32: Average feature importance for original and generated variables for Decision Trees (left), Random Forests (middle) and Gradient Boosting (right).

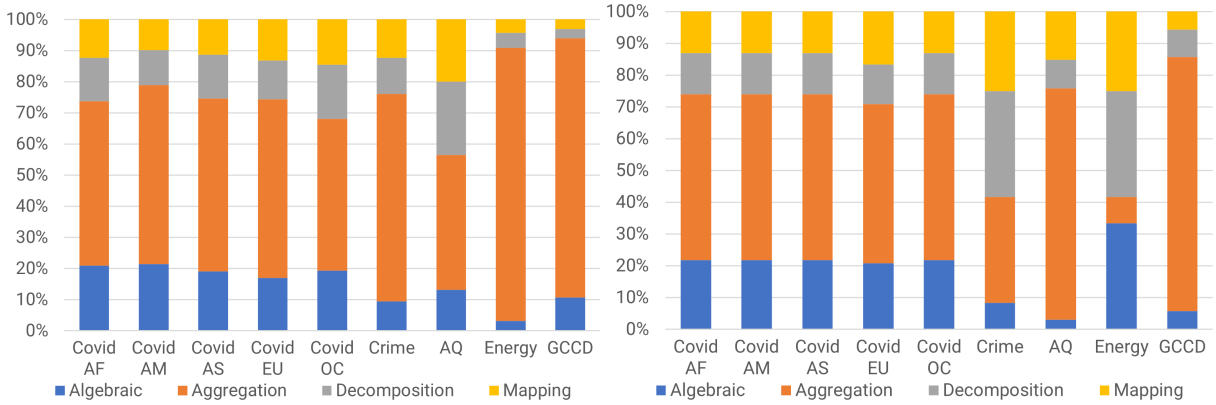

Figure 4.33: Time comparison (left) and amount of generated features (right) per type of operation.

models, as seen in the preprocessed baseline (`Base + Prep`), which depending on the model can have better AUC than the results from running DANKFE-II. However, adding these techniques to DANKFE-II also results in more robust models, with higher scores than the preprocessed baseline for all models, especially in Naive Bayes with an increase of over 10 pp. Compared to auto-sklearn, DANKFE-III achieves a higher score than the AutoML framework in all models expect Naive Bayes, with the highest score being Gradient Boosting with an increase of 5.3 pp.

In terms of time spent per record, DANKFE-III has a small increase compared to the previous iteration. This is due to the larger amount of variables generated by automatic operations, such as decomposition of dates and summaries of numeric variables, as well as preprocessing techniques. Fig. 4.31 shows that these values stay constant independently of the model used, similar to before, at around 50 milliseconds per record, which is a substantial decrease from the 300 milliseconds by auto-sklearn, while also achieving stronger results.

Similarly to DANKFE-II, all models give larger importance to generated features instead of the original ones, except for Decision Trees and Gradient Boosting in the GCCD dataset and Random Forests in the AQ dataset, which can be seen in fig. 4.32.

Compared to DANKFE-II, all datasets have more aggregation-based variables due to the automatic generation of the summary for numeric variables, which can be seen on fig. 4.33 (right). The larger amount of these variables contributes to a larger time spent in generating them, which can be seen in

fig. 4.33 (left).

Finally, to test how the DANKFE algorithm works in terms of scalability, different samples of a larger GCCD dataset (containing 135k records) were taken with different sizes, generating the same set of variables for each one.

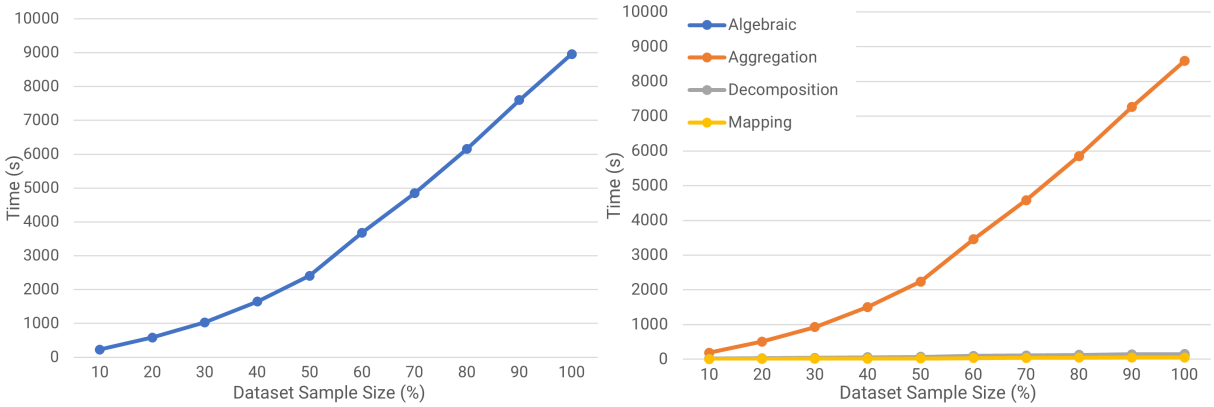

Figure 4.34: Scalability study: total time on variable generation (left) per types of variables generated (right).

Fig. 4.34 (left) shows the total amount of time spent on feature generation when varying the dataset size, which includes the time spent on reading the original dataset and writing the extended version (negligible compared to the time spent generating the features). It is clear that the algorithm presents a linear growth in time depending on the number of records. This behavior can be further detailed in fig. 4.34 (right), where we can see that compared to aggregation-based operations, record-based ones (algebraic, decomposition and mapping) take residual time. This evidences the trade-off between the different iterations of DANKFE, where processing speed (and therefore time to put a model into production, as per the KDD process) is sacrificed to give room for more functionality and therefore models with better performance, using potentially interesting variables that require aggregation-based operations.

5

Conclusion

Contents

In this chapter, we present an overall conclusion of the work proposed in chapter 3, reviewing its advantages and disadvantages and future work to further develop the DANKFE system.

5.1 Conclusions

In this era of Big Data, where the amount of data that needs to be processed on a daily basis is becoming intractable for the amount of data scientists that are qualified for a specific domain, the paradigm of data science is being shifted, where the focus is not on creating the best possible model for a specific problem, but rather to make machine learning more available to the general public, while keeping a good performance, in order to still make the most of the benefits that come from applying the KDD process (and its industrialized versions) into the data required by companies.

This paradigm is leading to the rapid development of AutoML frameworks, with black-box domain agnostic approaches that achieve good results without the need for expert data scientists (which can lead to some mistrust in such frameworks), but do not take any advantage of domain knowledge in the data preparation step, which is known to improve results and the amount of useful information retrievable from a system, since it is the most demanding part of the workflow.

Therefore, we proposed a system that extends datasets to increase the amount of information using domain knowledge stored in an ER model. Following the KDD process, we implemented an algorithm that is able to interpret the knowledge in the ER diagrams and extend a given dataset, using a set of operations (record-based, aggregation-based and/or a composition of both) stored inside a Production Rule System inside the algorithm to augment the feature space and possibly improve the performance of machine learning models trained with that dataset. The algorithm works for every domain as long as there is domain knowledge present in an ER model, which is widely used in the database community. Even if there is not much domain knowledge available, the DANKFE algorithm can be coupled to the rest of the data preparation and feature engineering phase of the data science workflow by automatically generating some potentially interesting variables, as well as cleaning, scaling and balancing the dataset.

Following the KDD process methodology for evaluation, a case study composed of several datasets of different domains was created, as well as a classification problem and an ER model with some domain knowledge for every dataset. Validating the algorithm in both efficacy and efficiency, results show that feature generation through the use of domain knowledge improves the quality of models, for all machine learning models tested (varying in simplicity). Not only does the score improve, but also the time spent in generation and training remains short, with DANKFE-III spending 50 milliseconds per record on average. Compared to a popular AutoML framework (auto-sklearn), the generation of features yielded better results except for Naive Bayes, and while taking at worse around 6x less time per record.

In conclusion, making use of domain knowledge could prove very beneficial for automatic machine

learning methods which currently do not make use of it, since it is able to improve model performance with low added complexity. It also gives data scientists a way to continue to use their valuable domain knowledge in the implementation of their models, which could improve the trust in these systems.

5.2 System Limitations and Future Work

While the DANKFE system shows strong results compared to a recent and widely used AutoML approach, it also has some limitations, as previously stated. The system is currently limited to a set of defined operations that while span a large amount of possibilities in terms of feature generation, it is not yet possible to generate every single kind of feature. For this to be possible, the DANKFE system would require a decoder that would translate automatically the operations inside the ER model into Python functions, before running them.

Even though it was not the main focus of our work, the data preparation surrounding the DANKFE algorithm could also be improved, by enlarging the amount of preprocessing operations that a user can choose to perform on the data, such as dummification, type transformations or other kinds of encoding. While results show that the generated features tend to be considered useful by ML models, it is also known that the presence of redundant or irrelevant features can hurt models, as well as add unnecessary temporal and spatial complexity. To mitigate this, techniques such as feature selection or extraction, which are part of feature engineering, as seen in section 2.1.2, could also be tested, ensuring that only relevant or important features are added to the input dataset.

While DANKFE runs much faster than an AutoML approach, it can still be further optimized, by generating multiple features at the same time, given that the features follow the same constraints. The current implementation of DANKFE only processes one feature at a time, which can be slow if the dataset is large and especially the generated features use aggregation-based variables. Lastly, other knowledge representations can be tested in the future, such as ontologies or extended ER models [87], which have the benefits of allowing axioms and inheritances, aggregations and compositions, respectively. While ER models are more publicly available, these other approaches could allow for more functionalities.

Bibliography

[1] S. Markovitch and D. Rosenstein, "Feature generation using general constructor functions," *Machine Learning*, vol. 49, pp. 59–98, 2004.

[2] J. Waring, C. Lindvall, and R. Umeton, "Automated machine learning: Review of the state-of-the-art and opportunities for healthcare," *Artificial Intelligence in Medicine*, vol. 104, p. 101822, 2020.

[3] C. J. Matheus, "The need for constructive induction," in *Machine Learning Proc. 1991*. Elsevier, 1991, pp. 173–177.

[4] Y. Terziev, "Feature generation using ontologies during induction of decision trees on linked data." in *DC@ ISWC*, 2016, pp. 90–98.

[5] S. Miller and D. Hughes, "The quant crunch: How the demand for data science skills is disrupting the job market," *Burning Glass Technologies*, 2017.

[6] D. Wang, J. D. Weisz, M. Muller, P. Ram, W. Geyer, C. Dugan, Y. Tausczik, H. Samulowitz, and A. Gray, "Human-ai collaboration in data science: Exploring data scientists' perceptions of automated ai," *Proc. ACM on Human-Computer Interaction*, vol. 3, no. CSCW, pp. 1–24, 2019.

[7] C. Antunes and A. Silva, "New trends in knowledge driven data mining." in *ICEIS (1)*, 2014, pp. 346–351.

[8] M. Feurer, K. Eggensperger, S. Falkner, M. Lindauer, and F. Hutter, "Auto-sklearn 2.0: Hands-free automl via meta-learning," *arXiv preprint arXiv:2007.04074*, 2020.

[9] U. Fayyad, G. Piatetsky-Shapiro, and P. Smyth, "From data mining to knowledge discovery in databases," *AI magazine*, vol. 17, no. 3, pp. 37–37, 1996.

[10] W. S. Cleveland, "Data science: An action plan for expanding the technical areas of the field of statistics," *International Statistical Review / Revue Internationale de Statistique*, vol. 69, no. 1, pp. 21–26, 2001. [Online]. Available: http://www.jstor.org/stable/1403527

[11] G. Piatetsky-Shapiro, “Crisp-dm, still the top methodology for analytics, data mining, or data science projects,” Oct 2014. [Online]. Available: https://www.kdnuggets.com/2014/10/crisp-dm-top-methodology-analytics-data-mining-data-science-projects.html

[12] P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. P. Reinartz, C. Shearer, and R. Wirth, “Crisp-dm 1.0: Step-by-step data mining guide,” 2000.

[13] N. Hotz, “What is crisp dm?” Apr 2022. [Online]. Available: https://www.datascience-pm.com/crisp-dm-2/

[14] IBM, “Analytics solutions unified method implementations with agile principles,” Mar 2016.

[15] S. Institute, “Introduction to semma,” Aug 2017. [Online]. Available: https://documentation.sas.com/doc/en/emref/14.3/n061bzurmej4j3n1jnj8bbjjm1a2.htm

[16] A. Azevedo and M. F. Santos, “Kdd, semma and crisp-dm: a parallel overview,” in *IADIS European Conf. Data Mining*, 2008.

[17] P. M. Domingos, “A few useful things to know about machine learning,” *Communications of the ACM*, vol. 55, pp. 78 – 87, 2012.

[18] H. Liu and H. Motoda, *Feature selection for knowledge discovery and data mining*, 1st ed. Springer US, 1998.

[19] T. Hastie, J. Friedman, and R. Tisbshirani, *High-Dimensional Problems*, 2nd ed. Springer, 2017, p. 649–650.

[20] H. Liu and H. Motoda, *Feature extraction, construction and selection: A data mining perspective*. Springer Science & Business Media, 1998, vol. 453.

[21] S. Donoho and L. Rendell, “Feature construction using fragmentary knowledge,” in *Feature Extraction, Construction and Selection*. Springer, 1998, pp. 273–288.

[22] R. Brachman, R. Levesque, H. Levesque, and M. Pagnucco, *Knowledge Representation and Reasoning*, ser. The Morgan Kaufmann Artificial. Elsevier Science, 2004. [Online]. Available: https://books.google.pt/books?id=OuPtLaA5QjoC

[23] R. Davis, H. Shrobe, and P. Szolovits, “What is a knowledge representation?” *AI magazine*, vol. 14, no. 1, pp. 17–17, 1993.

[24] P. Domingos and D. Lowd, “Unifying logical and statistical ai with markov logic,” *Communications of the ACM*, vol. 62, no. 7, pp. 74–83, 2019.

[25] K. Trentelman, “Survey of knowledge representation and reasoning systems,” 2009.

[26] J. Lambek and P. Scott, *Introduction to Higher-Order Categorical Logic*, ser. Cambridge Studies in Advanced Mathematics. Cambridge University Press, 1988. [Online]. Available: https://books.google.pt/books?id=6PY_emBeGjUC

[27] M. Minsky, *A framework for representing knowledge*. de Gruyter, 2019.

[28] J. Sowa, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Elsevier Science, 2014. [Online]. Available: https://books.google.pt/books?id=lTKnCQAAQBAJ

[29] ——, *Knowledge Representation: Logical, Philosophical, and Computational Foundations*. Brooks/Cole Publishing Co., 01 2000.

[30] B. Chandrasekaran, J. Josephson, and V. Benjamins, "What are ontologies, and why do we need them?" *IEEE Intelligent Systems*, vol. 14, no. 1, pp. 20–26, Jan. 1999. [Online]. Available: http://ieeexplore.ieee.org/document/747902/

[31] H. Cešpivová, J. Rauch, V. Svatek, M. Kejkula, and M. Tomeckova, "Roles of medical ontology in association mining crisp-dm cycle," in *ECML/PKDD04 Workshop on Knowledge Discovery and Ontologies (KDO'04), Pisa*, vol. 220. Citeseer, 2004.

[32] M. G. Taylor, K. Stoffel, and J. A. Hendler, "Ontology-based induction of high level classification rules." in *DMKD*. Citeseer, 1997, p. 0.

[33] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific american*, vol. 284, no. 5, pp. 34–43, 2001.

[34] N. Shadbolt, T. Berners-Lee, and W. Hall, "The semantic web revisited," *IEEE intelligent systems*, vol. 21, no. 3, pp. 96–101, 2006.

[35] P. Hitzler, "A review of the semantic web field," *Communications of the ACM*, vol. 64, no. 2, pp. 76–83, 2021.

[36] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowledge Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993. [Online]. Available: https://linkinghub.elsevier.com/retrieve/pii/S1042814383710083

[37] P. P.-S. Chen, "The entity-relationship model: Toward a unified view of data," *ACM Transactions on Database Systems*, 1976.

[38] N. Friedman, D. Geiger, and M. Goldszmidt, "Bayesian network classifiers," *Machine learning*, vol. 29, no. 2, pp. 131–163, 1997.

[39] S. Flügge, S. Zimmer, and U. Petersohn, “Knowledge representation and diagnostic inference using bayesian networks in the medical discourse,” *arXiv preprint arXiv:1909.08549*, 2019.

[40] M. Richardson and P. Domingos, “Markov logic networks,” *Machine learning*, vol. 62, no. 1-2, pp. 107–136, 2006.

[41] U. Khurana, D. Turaga, H. Samulowitz, and S. Parthasrathy, “Cognito: Automated feature engineering for supervised learning,” in *2016 IEEE 16th International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2016, pp. 1304–1307.

[42] G. Pagallo and D. Haussler, “Boolean feature discovery in empirical learning,” *Machine learning*, vol. 5, no. 1, pp. 71–99, 1990.

[43] P. M. Murphy and M. J. Pazzani, “Id2-of-3: Constructive induction of m-of-n concepts for discriminators in decision trees,” in *Machine Learning Proc. 1991*. Elsevier, 1991, pp. 183–187.

[44] Z. Zheng, “Constructing x-of-n attributes for decision tree learning,” *Machine learning*, vol. 40, no. 1, pp. 35–75, 2000.

[45] H. Ragavan and L. A. Rendell, “Lookahead feature construction for learning hard concepts,” in *ICML*, 1993.

[46] Y.-J. Hu and D. Kibler, “Generation of attributes for learning algorithms,” in *AAAI/IAAI, Vol. 1*, 1996, pp. 806–811.

[47] W. Fan, E. Zhong, J. Peng, O. Verscheure, K. Zhang, J. Ren, R. Yan, and Q. Yang, “Generalized and Heuristic-Free Feature Construction for Improved Accuracy,” in *Proc 2010 SIAM International Conference on Data Mining*. Society for Industrial and Applied Mathematics, Apr. 2010, pp. 629–640. [Online]. Available: https://epubs.siam.org/doi/10.1137/1.9781611972801.55

[48] J. M. Kanter and K. Veeramachaneni, “Deep feature synthesis: Towards automating data science endeavors,” in *2015 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*. Campus des Cordeliers, Paris, France: IEEE, Oct. 2015, pp. 1–10. [Online]. Available: http://ieeexplore.ieee.org/document/7344858/

[49] G. Katz, E. C. R. Shin, and D. Song, “ExploreKit: Automatic Feature Generation and Selection,” in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. Barcelona, Spain: IEEE, Dec. 2016, pp. 979–984. [Online]. Available: http://ieeexplore.ieee.org/document/7837936/

[50] H. T. Lam, J.-M. Thiebaut, M. Sinn, B. Chen, T. Mai, and O. Alkan, “One button machine for automating feature engineering in relational databases,” *arXiv:1706.00327 [cs]*, Jun. 2017, arXiv: 1706.00327. [Online]. Available: http://arxiv.org/abs/1706.00327

[51] A. Kaul, S. Maheshwary, and V. Pudi, "AutoLearn — Automated Feature Generation and Selection," in *2017 IEEE International Conference on Data Mining (ICDM)*. New Orleans, LA: IEEE, Nov. 2017, pp. 217–226. [Online]. Available: http://ieeexplore.ieee.org/document/8215494/

[52] F. Nargesian, H. Samulowitz, U. Khurana, E. B. Khalil, and D. S. Turaga, "Learning feature engineering for classification." in *Ijcai*, 2017, pp. 2529–2535.

[53] U. Khurana and H. Samulowitz, "Autonomous predictive modeling via reinforcement learning," in *Proc. 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3285–3288.

[54] X. Chen, Q. Lin, C. Luo, X. Li, H. Zhang, Y. Xu, Y. Dang, K. Sui, X. Zhang, B. Qiao *et al.*, "Neural feature search: A neural architecture for automated feature engineering," in *2019 IEEE International Conference on Data Mining (ICDM)*. IEEE, 2019, pp. 71–80.

[55] Y. Xie, Z. Wang, Y. Li, B. Ding, N. M. Gürel, C. Zhang, M. Huang, W. Lin, and J. Zhou, "Fives: Feature interaction via edge search for large-scale tabular data," 2021.

[56] K. Krawiec, "Genetic programming-based construction of features for machine learning and knowledge discovery tasks," *Genetic Programming and Evolvable Machines*, vol. 3, no. 4, pp. 329–343, 2002.

[57] M. G. Smith and L. Bull, "Genetic programming with a genetic algorithm for feature construction and selection," *Genetic Programming and Evolvable Machines*, vol. 6, no. 3, pp. 265–281, 2005.

[58] B. Tran, B. Xue, and M. Zhang, "Genetic programming for multiple-feature construction on high-dimensional classification," *Pattern Recognition*, vol. 93, pp. 404–417, 2019.

[59] J. Ma and X. Gao, "A filter-based feature construction and feature selection approach for classification using genetic programming," *Knowledge-Based Systems*, vol. 196, p. 105806, 2020.

[60] F. Hutter, L. Kotthoff, and J. Vanschoren, Eds., *Automated Machine Learning: Methods, Systems, Challenges*, ser. The Springer Series on Challenges in Machine Learning. Cham: Springer International Publishing, 2019. [Online]. Available: http://link.springer.com/10.1007/978-3-030-05318-5

[61] R. Elshawi, M. Maher, and S. Sakr, "Automated Machine Learning: State-of-The-Art and Open Challenges," *arXiv:1906.02287 [cs, stat]*, Jun. 2019, arXiv: 1906.02287. [Online]. Available: http://arxiv.org/abs/1906.02287

[62] X. He, K. Zhao, and X. Chu, "AutoML: A Survey of the State-of-the-Art," *Knowledge-Based Systems*, vol. 212, p. 106622, Jan. 2021, arXiv: 1908.00709. [Online]. Available: http://arxiv.org/abs/1908.00709

[63] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined Selection and Hyperparameter Optimization of Classification Algorithms," *arXiv:1208.3719 [cs]*, Mar. 2013, arXiv: 1208.3719. [Online]. Available: http://arxiv.org/abs/1208.3719

[64] H. Jin, Q. Song, and X. Hu, "Auto-Keras: An Efficient Neural Architecture Search System," *arXiv:1806.10282 [cs, stat]*, Mar. 2019, arXiv: 1806.10282. [Online]. Available: http://arxiv.org/abs/1806.10282

[65] L. Zimmer, M. Lindauer, and F. Hutter, "Auto-Pytorch: Multi-Fidelity MetaLearning for Efficient and Robust AutoDL," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 3079–3090, Sep. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9382913/

[66] N. Erickson, J. Mueller, A. Shirkov, H. Zhang, P. Larroy, M. Li, and A. Smola, "Autogluon-tabular: Robust and accurate automl for structured data," *arXiv preprint arXiv:2003.06505*, 2020.

[67] E. LeDell and S. Poirier, "H2o automl: Scalable automatic machine learning," in *Proc. AutoML Workshop at ICML*, vol. 2020, 2020.

[68] R. S. Olson and J. H. Moore, "Tpot: A tree-based pipeline optimization tool for automating machine learning," in *Workshop on automatic machine learning*. PMLR, 2016, pp. 66–74.

[69] D. W. Aha, P. Clark, S. Salzberg, G. Blix, and R. B. F. P. Newld, "Incremental constructive induction: An instance-based approach," 1991.

[70] J. M. Aronis and F. J. Provost, "Efficiently constructing relational features from background knowledge for inductive machine learning." in *KDD Workshop*, 1994, pp. 347–358.

[71] M. Badian and S. Markovitch, "Knowledge-Based Learning through Feature Generation," *arXiv:2006.03874 [cs, stat]*, Jun. 2020, arXiv: 2006.03874. [Online]. Available: http://arxiv.org/abs/2006.03874

[72] E. Gabrilovich and S. Markovitch, "Wikipedia-based semantic interpretation for natural language processing," *Journal of Artificial Intelligence Research*, vol. 34, pp. 443–498, 2009.

[73] P. Wang, J. Hu, H.-J. Zeng, and Z. Chen, "Using wikipedia knowledge to improve text classification," *Knowledge and Information Systems*, vol. 19, no. 3, pp. 265–281, 2009.

[74] X. Hu, X. Zhang, C. Lu, E. K. Park, and X. Zhou, "Exploiting wikipedia as external knowledge for document clustering," in *Proc. 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2009, pp. 389–396.

[75] L. Zhang, C. Li, J. Liu, and H. Wang, “Graph-based text similarity measurement by exploiting wikipedia as background knowledge,” *World Academy of Science, Engineering and Technology*, vol. 59, pp. 1548–1553, 2011.

[76] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.

[77] W. Cheng, G. Kasneci, T. Graepel, D. Stern, and R. Herbrich, “Automated feature generation from structured knowledge,” in *Proc. 20th ACM international conference on Information and knowledge management*, 2011, pp. 1395–1404.

[78] H. Paulheim, P. Ristoski, E. Mitichkin, and C. Bizer, “Data mining with background knowledge from the web,” *RapidMiner World*, pp. 1–14, 2014.

[79] T. Tim Berners-Lee, “Linked data,” Jul 2006. [Online]. Available: https://www.w3.org/DesignIssues/LinkedData.html

[80] S. Kramer, N. Lavrač, and P. Flach, “Propositionalization approaches to relational data mining,” *Relational data mining*, pp. 262–291, 2001.

[81] S. Galhotra, U. Khurana, O. Hassanzadeh, K. Srinivas, H. Samulowitz, and M. Qi, “Automated feature enhancement for predictive modeling using external knowledge,” in *2019 International Conference on Data Mining Workshops (ICDMW)*. IEEE, 2019, pp. 1094–1097.

[82] S. Bloehdorn and A. Hotho, “Boosting for text classification with semantic features,” in *International workshop on knowledge discovery on the web*. Springer, 2004, pp. 149–166.

[83] L. Friedman and S. Markovitch, “Recursive feature generation for knowledge-based learning,” *arXiv preprint arXiv:1802.00050*, 2018.

[84] A. G. Salguero, J. Medina, P. Delatorre, and M. Espinilla, “Methodology for improving classification accuracy using ontologies: application in the recognition of activities of daily living,” *Journal of Ambient Intelligence and Humanized Computing*, vol. 10, no. 6, pp. 2125–2142, 2019.

[85] J. McCarthy, “Generality in artificial intelligence,” *Communications of the ACM*, vol. 30, no. 12, pp. 1030–1035, 1987.

[86] T. Fawcett, “An introduction to roc analysis,” *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[87] R. Elmasri and S. B. Navathe, *The Enhanced Entity–Relationship (EER) Model*. Addison-Wesley, 2000, p. 107–135.